EXPLOITING STRUCTURES OF TRAJECTORY OPTIMIZATION FOR EFFICIENT OPTIMAL MOTION PLANNING

Draft of March 12, 2021 at 15 : 33

BY

GAO TANG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Doctoral Committee:

Professor Kris Hauser, Chair
Professor David Forsyth
Professor Saurabh Gupta
Professor Girish Chowdhary
Professor Ludovic Righetti

ABSTRACT

Trajectory optimization is an important tool for optimal motion planning due to its flexibility in cost design, capability to handle complex constraints, and optimality certification. It has been widely used in robotic applications such as autonomous vehicles, unmanned aerial vehicles, humanoid robots, and highly agile robots. However, practical robotic applications often possess nonlinear dynamics and non-convex constraints and cost functions, which makes the trajectory optimization problem usually difficult to be efficiently solved to global optimum. The long computation time, possibility of non-convergence, and existence of local optima impose significant challenges to applying trajectory optimization in reactive tasks with requirements of real-time replanning. In this thesis, two structures of optimization problems are exploited to significantly improve the efficiency, i.e. computation time, reliability, i.e. success rate, and optimality, i.e. quality of the solution. The first structure is the existence of a convex sub-problem, i.e. the problem becomes convex if a subset of optimization variables is fixed and removed from the optimization. This structure exists in a wide variety of problems, especially where decomposition of spatial and temporal variables may result in convex sub-problem. A bilevel optimization framework is proposed that optimizes the subset and its complement hierarchically where the upper level optimizes the subset with convex constraints and the lower level uses convex optimization to solve its complement. The key is to use the solution of the lower level problem to compute analytic gradients for the upper level problem. The bilevel framework is reliable due to its convex lower problem, efficient due to its simple upper problem, and yields better solutions than alternatives, although the existence requirement of convex sub-problem is generally too strict for many applications. The second structure is the local continuity of the argmin function for parametric optimization problems which map from problem parameters to the corresponding optimal solutions. The argmin function can be approximated from data which is collected offline by sampling the problem parameters and solving them to optima. Three approaches are proposed to learn the argmin from data each suited best for distinct applications. The nearest-neighbor optimal control searches problems with similar parameters and uses their solution to initialize nonlinear optimization. For problems with globally continuous argmin, neural networks can be used to learn from data and a few steps of convex optimization can further improve their predictions. As for problems with discontinuous argmin, mixture of experts (MoE) models are used. The MoE contains several experts and a classifier and is trained by splitting the data first according to discontinuity of argmin and then training each

expert independently. Both empirical k-Means and theoretical topological data analysis approaches are explored for discontinuity identification and finding suitable data splits. Both methods result in data splits that help train MoE models that outperform the discontinuity-agnostic learning pipeline using standard neural networks. The trajectory learning approach is efficient since it only requires model evaluation to compute a trajectory, reliable since the MoE model is accurate after correctly handling discontinuity, and optimal since the data are collected offline and solved to optimal. Moreover, this local continuity structure is less restrictive and exists for a wide range of non-degenerate problems. Exploitation of these two structures helps build an efficient optimal motion planner with high reliability.

"To my family, for their love and support."

ACKNOWLEDGMENTS

Time flies and my journey of pursing a PhD is about to end. Thinking back to the past five years, I cannot help asking myself: "what if you had not been fortunate enough to meet all the people who have helped you?"

I would first like to thank my advisor, Professor Kris Hauser, without whom this fantastic journey would never have existed. His expertise in the robotics field and wide range of knowledge helped me to jump out of my comfort zone and explore unheard methods. His valuable suggestions during our weekly meetings guided me through this journey and brought my work to a higher level. The freedom he gave allowed me to explore research directions that interest me and there were countless times that he correctly pulled me back on the right track before I wasted too much time on my immature ideas. I cannot thank him enough for all the comments in paper writings and presentation preparations as well.

I would also like to thank other professors that helped me with my research during collaborations. I want to thank Professor Joseph Izatt and Dr. Anthony Kuo at Duke for their support and insightful suggestions in our eye surgery meetings. I want to thank Professor Leila Bridgeman at Duke for her instructions to help me understand model predictive control in a theoretical way. I also appreciate the useful instructions from Professor Todd Murphey from Northwestern in the autompc project.

I have to thank my co-authors with whom research has never been so easy. I would like to thank Weidong (Bill) Sun at Duke for his assistance in quadcopter physical experiments, and collaborations in brainstorming ideas, running experiments, writing papers, and conference presentations. I would like to thank Yuan Tian at Duke for his assistance in physical experiments for the robotic eye surgery project. I could never have been involved in medical research without the collaborations with Mark Draelos, Brenton Keller, and Yuan Tian. I have to thank William Edwards for his massive contributions in the autompc and eye surgery project. I appreciate the insightful ideas from Giorgos Mamakoukas in our autompc discussions.

I would like to thank my colleagues and lab-mates for not only academic assistance but also after-work activities. I shall never forget the help I got from and happiness I had with my lab-mates–Jianqiao Li, Hayden Bader, Adam Konneker, Fan Wang, Haibei Zhu, Shihao Wang, Wuming Zhang, Yilun Zhou, Yifan Zhu, João Marques, William Edwards, Amnon Attali, Mengchao Zhang, Jaejun Park, Hyunjik Park, Yeonju Kim, Patrick Naughton, Yu Zhou, and Zherong Pan. It is fortunate that I had the chance to work with you in the same

lab.

Last but not least, I would like to thank my family for their support in pursing this degree. It will be a pain forever in my heart that we haven't met for five years. I have to thank my climbing friends without whom this journey would be dull–Wen Zhou, Jingjing Wang, Xiaoyang Yao, Yingru Xu, Kris Hauser, Amnon Attali, Patrick Naughton and Yu Zhou. Finally I have to thank my close friends for the memorable moments I had with Xing He, Ruiyi Zhang, Jingxuan Ding, Yan Zhang, Yuanjun Yao, Weidong Sun, Yifan Zhu, Mengchao Zhang, Jiaqi Guan, and Haoran Qiu.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Motion planning is an essential component in autonomous systems as it computes how a system should move to accomplish its tasks. There are a few desirable properties for motion planners:

- Being able to handle different task specifications such as having different start and goal states, working on new environments, and imposing different constraints on the motion.
- Being able to exploit potentially underactuated and weakly actuated system dynamics for aggressive behavior and long-term planning.
- Efficiency – so they may compute a plan within a limited time budget for reactive tasks.
- Reliability – so they do not fail to compute a plan for feasible tasks in order to minimize task termination and human intervention.
- Being able to compute optimal plans for various objective functions such as task time to increase task efficiency, control cost to conserve energy or risk probability to maximize safety.

However, it is challenging to have a planner that satisfies all those requirements, especially for systems with nonlinear dynamics, underactuation, and complex constraints [1, 2, 3]. A potential candidate is trajectory optimization which can handle nonlinear dynamics, underactuation, complex constraints, and different objective functions [1].

This class of motion planners is the focus of this work, and the readers are referred to [2] for an introduction to other types of motion planners not covered here. In this chapter, the formulation of trajectory optimization, previous research on this topic, typical methods to solve it, trends in recent work, and existing open problems are briefly introduced. Two types of structures in trajectory optimization problems and methods to exploit them in this thesis are also introduced.

1.1 BACKGROUND

Trajectory optimization originates from optimal control theory, which deals with finding a control sequence for a dynamical system over a period of time such that an objective

function is optimized subject to constraints. For a system governed by dynamical equation $\dot{x} = f(x, u)$, the trajectory optimization computes the control trajectory $u(t) : [0, T] \to \mathcal{U} \subseteq \mathbb{R}^m$ and state trajectory $x(t) : [0, T] \to \mathcal{X} \subseteq \mathbb{R}^n$ that minimizes some measure of performance J while satisfying all the necessary constraints, e.g. being collision-free and dynamically feasible, where m and n are the dimension of the control and state space, respectively. This optimization is formulated as

$$\begin{aligned} \underset{x,u,T}{\text{minimize}} \quad & J(x, u, T) = \int_0^T \ell(x, u, t)dt + \Phi(x(T)) \\ \text{subject to} \quad & \dot{x} = f(x, u) \\ & x(0) \in \mathcal{X}_0, x(T) \in \mathcal{X}_f \\ & x(t) \in \mathcal{X}, u(t) \in \mathcal{U} \quad \forall t \in [0, T] \\ & \phi(x, u) \leq 0, \quad \forall t \in [0, T] \end{aligned} \tag{1.1}$$

where $T > 0$ is the potentially unknown execution time; the system starts from initial state set $\mathcal{X}_0$ and ends at terminal state set $\mathcal{X}_f$; ϕ denotes the additional path constraints on the state and control; $\ell(x, u, t)$ is the running cost and $\Phi(x(T))$ is the terminal cost. The flexibility in the design of performance indices and the inclusion of constraints allows us to obtain desired system behaviors such as aggressive motion to minimize task time, momentum gaining to handle weak actuation, low-jerk motion to minimize control effort, and safety-prioritized trajectory to minimize risks. Note that we use $x \equiv x(t)$ and $u \equiv u(t)$ for simplicity throughout the thesis when there is no ambiguity. Following [4, 5], a brief history of previous efforts into solving the trajectory optimization problem in Eq. (1.1) is introduced.

1.1.1 Calculus of Variations

Optimal control is an important application of the Calculus of Variations (CV). The history of CV dates back to the 17th century when the "branchistochrone" problem was posed by Galileo Gallilei (1564-1642) which was later solved by John Bernoulli, Leibniz, and Isaac Newton using CV. Leonard Euler (1707-1783) and Jean Louis Lagrange (1736-1813) derived the first-order necessary conditions for a optimal solution, called Euler-Lagrange equations today. Many researchers contributed to the theory of CV, the most important of which is Pontryagin (1908-1988) for developing the "maximum principle" [6] that deals with the presence of constraints for the state and input controls.

Most optimal control problems can be solved via the Euler-Lagrange equations and the maximum principle. For introductory purposes, let us consider a simple problem with fixed

start point, fixed duration,T, terminal constraint, constrained control, and no path constraints. One solves this problem by first introducing Lagrangian multipliers (sometimes also called costates) $\lambda(t)$ associated with the constraint of system dynamics and ν associated with the terminal constraint. Then the Euler-Lagrange equations are applied so one derives the evolution of $\lambda(t)$ as an Ordinary Differential Equation (ODE). The optimal control is derived using Pontryagin's maximum principle which is usually a function of state x and multiplier λ subject to constraints on control. The terminal state and ν are derived to satisfy some equations called *transversality conditions*.

The complete set of necessary conditions consists of the ODE of system dynamics, the ODE of $\lambda(t)$, and transversality conditions at $t = T$. This is often referred to as a *two point boundary value problem* (TPBVP). The readers are referred to [6] for more general cases such as free T and path constraints. This TPBVP is usually solved with shooting methods. One starts with some guess of the initial value of $\lambda(t)$, i.e. $\lambda(0)$, integrates the two ODEs involving the evolution of state $x(t)$ and Lagrangian multipliers $\lambda(t)$, and check if the boundary conditions at $t = T$ are satisfied. The goal of shooting methods is to find $\lambda(0)$ such that boundary conditions at $t = T$ are all satisfied. This is well solved by root-finding algorithms and once $\lambda(0)$ is found, the optimal trajectory is obtained by integrating the ODEs. For specific problems, closed-form solutions can be obtained. One of the most important applications of this method is the *Linear Quadratic Regulator* or *LQR* by Kalman [7] in 1960 where the system's dynamics are linear and the objective function is quadratic. For general nonlinear problems, numerical methods are used to find these roots.

This type of approach to find the optimal trajectory by CV and root-finding algorithms is usually called an *indirect method*. Indirect methods were first widely used in aerospace engineering such as spacecrafts [8], rockets [6, 9], and aircrafts [10]. Indirect methods are usually efficient due to the small amount of unknowns, and fast convergence rate of nonlinear equation solver near its solution. The main issue of indirect methods is the difficulty in finding the root of the TPBVP. In the presence of strong nonlinearities, the domain of convergence of root-finding algorithms may be too narrow to converge from a random initial guess. Moreover, the unknown Lagrange multipliers have no physical meaning and it is, thus, often difficult to provide a reasonable initial guess for their values. The necessary complex algebraic manipulation is another hurdle to automating indirect methods. One of the most fatal shortcoming that limits its application area is the difficulty in handling inequality constraints, especially inequality path constraints of states which exist in most robotic applications such as collision avoidance. As a result, indirect methods are often used in applications with simple constraints [11, 12] despite efforts in [13, 14] to handle complex path constraints with limited success.

Since the theory of optimal control has been long established and there is not much flexibility in TPBVP derivation, the research on indirect methods is often focused on

- How to improve the success rate and efficiency for specific problems [12, 15, 16]
- how to apply it to more challenging problems [17]
- how to provide initial guesses via simplification [14], and combination with other methods [18].

These methods, however, require specific domain knowledge, are problem specific and difficult to be generalized to other problems.

In this thesis, a general framework to provide initial guesses for indirect methods is proposed. It uses the assumption that similar problems have similar solutions. For instance, the optimal control problem starting from two close starts should have similar initial costates if everything else is the same. This idea motivates to parameterize the problem of interest, sample problem parameters, precompute a dataset of (problem, solution) pairs, and approximate the mapping from problem to solution from data. The framework, *Nearest Neighbor Optimal Control* (NNOC) is presented in Chapater 4 and shows orders of magnitude improvement in terms of computational efficiency.

1.1.2 Dynamic Programming

Bellman developed the theory of dynamic programming in the 1950s [19]. An optimal value function $V(x)$ was defined as the performance index starting from state x, and proceeding optimally to a terminal state. The optimal control function $u(x)$ is derived using the value function $V(x)$ and the system's dynamics. The Hamilton–Jacobi–Bellman (HJB) partial differential equation [6] is hard to solve analytically and a numerical solution often requires a discretization of the state space, leading to the "curse of dimensionality".

Despite the difficulty of solving HJB equation, the Bellman equation expressed in discrete form has seen many applications. If the state space is limited to a region near a nominal optimal trajectory, the dynamic programming problem can be approximated by a linear quadratic (LQ) problem with linearized dynamics equation and quadratic approximation to the performance index. Such idea has inspired the differential dynamic programming (DDP) [20], iterative LQR [21], and iLQG [22] methods. These methods iteratively build a local LQ problem near current trajectory and update the trajectory using the solution from local LQ problems. These methods however, require tedious derivations to obtain the trajectory update rule, especially in constrained optimization problems, and requires an initial guess

close to optimum, similar to the numerical optimization methods introduced later. Besides, these methods are generally not efficient enough in real-time applications unless the initial guess is close to optimum. These methods get the same optimal trajectory with the numerical optimization methods described in the next section, but are less flexible in terms of choices of cost functions and constraints.

Reinforcement learning (RL) [23], is another approach to solving optimal control problems. Classical tabular methods, such as value iteration, policy iteration, and Q-learning [23] are all based on the Bellman equation. However, tabular methods face the "curse of dimensionality" problem and policy gradient [24] method is usually used for problems with high-dimensional state and action space. Note that the policy gradient variant of RL does not explicitly use the Bellman equation in theoretic derivation, but some implementations, such as actor-critic [25] method, use it to improve their algorithm's stability. The main advantage of RL methods is it does not require the modeling of the system which is difficult in many applications. However, RL methods are usually not sample-efficient and require many "rollouts" on the actual system, making it unsuitable in many applications where physical experiments are expensive or unsafe to do [26]. One could potentially train a RL agent within physical simulator but it brings the issue of reality gap [27].

In this thesis, RL is mostly not addressed, since all the trajectory optimization methods studied here require a dynamics model of the system. Model-based RL does exist, but the its pipeline is to first fit a dynamics model from data and then find a policy subject to the fitted model. It is thus unfair to compare RL which does not require a model with methods in this thesis which require a precise model of the system. Theoretically speaking, one can perform the policy search part using the known dynamics model, but this approach is not easier than direct trajectory optimization and has difficulty imposing constraints by handcrafting the reward function [28].

1.1.3 Numerical Optimization

With the development of constrained numerical optimization after WWII, in particular the work of Kuhn and Tucker [29], large scale NLP solvers have been developed [30, 31]. The narrow convergence domain and inability to handle complex constraints of indirect methods motivate researchers to find alternatives to it. An alternative is to discretize the trajectory by values at a finite number of time points using a collocation method [32, 33] or transcription method [34, 35, 36]. The discretized optimization problem is then solved by sparse NLP solvers [37]. This approach, called *direct methods*, can be applied without deriving the necessary conditions which are treated by the optimizer. This feature makes

the method appealing for complicated applications and writing automatic software for direct methods is much easier than indirect methods. These methods are also able to handle path inequalities more easily than indirect methods as well and are, thus, widely used in many applications ranging from industrial robots [38], Unmanned Aerial Vehicles (UAV) [39, 40, 41, 42], Autonomous Vehicles (AV) [43, 44], and legged robots [45, 46, 47].

One of the most important applications of trajectory optimization is Model Predictive Control (MPC) [48, 49] which solves a finite-horizon optimal control sequence at every time step, apply the first optimal control command to the system, and repeat the process. It originates from chemical engineering where efficiency of trajectory optimization is not an issue and has recently gathered the attention of the control community. The limited time budget of robotics control applications only allows MPC be applied in systems with quadratic costs, linear dynamics, and linear constraints, and limited planning horizon, i.e. small-scale convex problems For nonlinear problems, a common practice is to use trajectory optimization to compute an open-loop nominal trajectory, and applying MPC to track the nominal trajectory after linearization of system dynamics around it. Similar control schemes are also applicable when MPC is replaced with LQR or Proportional-Integral-Derivative (PID) controller. In either case, trajectory optimization is used to compute a nominal trajectory.

Another type of direct methods uses a novel parameterization of the trajectories instead of discretization. For instance, a trajectory can be parameterized as high order polynomials and the coefficients are optimized to minimize some cost function. For problems with differential flatness [50], the dynamics equations are satisfied by computing the controls using the states and their high order derivatives. The path constraints are satisfied by either placing a grid on the trajectory [50] or deliberate parameterization such as control points of Bézier curves [40]. The flexibility of trajectory parameterization is another reason for the popularity of direct methods which allow for the exploitation of domain knowledge to simplify problems, even though this is not generalizable.

The flexibility of direct methods comes with drawbacks, such as the limited expressiveness of parameterization, large amount of unknowns and constraints [12]. In some cases, polynomials of finite order are selected as parameterization and may lead to overly conservative trajectories [51]. Both direct transcription and direct collocation methods discretize the state and control trajectory and may lead to large amount of unknowns and constraints depending on the discretization resolution. There is a tradeoff between problem scale and accuracy of discretization. Most importantly, nonlinear dynamics of the system and non-convex constraints of the problem result in a non-convex optimization problem. Non-convex optimization still remains as an open-question so it is unclear how to solve them efficiently

and reliably without getting into bad local optima.

Some representative recent research in direct methods include

- Application of direct methods to challenging problems such as legged robots with unknown contact sequences [45].
- Lossless convexification of the optimization problem [52].
- Problem simplification using domain knowledge and novel parameterizations [50, 53].
- Software for general trajectory optimization [54, 55].

These methods, however, do not address the inefficiency caused by non-convex optimization for general problems where expert knowledge is not available.

Although the large scale and non-convexity are not an issue for non-reactive tasks, due to their offline computation and physical grounding providing reasonable initial guesses for the optimization variables, the efficient computation of trajectories remains an open question.

1.2 STRUCTURE EXPLOITATION

In this thesis, we push forward the application of trajectory optimization in reactive tasks by improving its efficiency and reliability. Specifically, we take advantage of the underlying structures of some optimization problems and design corresponding solvers that outperform a general solver, agnostic to such structures.

The first structure is the existence of convex subproblems. This structure is seen in problems where the optimization variables can be split into two groups and fixing one group results in a convex problem for the other one. A well-known approach to take advantage of this structure is to identify the fixed group, heuristically choose a value for it, and optimize the other group with convex optimization [50, 53]. The first two steps of this approach require domain knowledge. For instance, in [40, 50] fixing the time allocation to each segment of a spline trajectory allows the spline coefficients be optimized by Quadratic Programming (QP). In [53] fixing the path makes the time-optimal problem convex by using a novel parameterization. We propose a method that improves upon this approach by lifting the restriction of fixing one group of variables using domain knowledge and, instead, efficiently optimizing variables in both groups.

For reactive tasks, the distribution of the control task can be described by a probability distribution of some parameters that uniquely specify a task. These types of planning tasks are called *parametric tasks* in this thesis. For instance, a robot navigating in a room

has to finish tasks parameterized by the start and goal locations. For parametric tasks, the trajectory optimization problems are *parametric optimization problems* [56, 57] as well. The solution of a parametric optimization problem is a mapping from problem parameter to the corresponding optimum [56]. The second structure is the local continuity of the solution of parametric optimization problems called the *argmin* function in this thesis. For a parametric optimization problem, the argmin function maps from a parameter to the optimal solution of the corresponding optimization problem. This local continuity means that, for every parameter, its solution is similar to the solution of some neighboring parameter. This property motivates the precomputation of a dataset of sampled problems and learning from this dataset to help solve new problems. We propose a machine learning method that builds a model using precomputed data that directly predicts an approximate optimal trajectory with high reliability so numerical optimization can be avoided, or in the worst case scenario, be accelerated by better initial guesses.

1.2.1 Convex Subproblems

Here a subproblem means the resulting optimization problem in the remaining optimization variables after fixing a subset of them. Formally,for a general constrained optimization problem $P : \min_x f(x) \text{ s.t. } g(x) \leq 0$ with a decomposition of the optimization variable x into two non-empty subsets $x_u \subset x$ and its compliment $x_l = x \setminus x_u$, a subproblem is defined as $P(x_u) : \min_{x_l} f(x_u, x_l) \text{ s.t. } g(x_u, x_l) \leq 0$. If $P(x_u)$ is convex for every valid choice of x_u, we say that P has convex subproblems. Such type of structure has been widely used in UAVs [40, 50], legged robots [58], manipulators [59], and vehicles [60]. In their work, the main idea is to use heuristics or sampling to find x_u in the first step and convex optimization to find x_l in the second. Apparently, this two-step approach depends on the choice of x_u which is non-trivial to compute and heuristics may work poorly given the nonlinearity of the problem. These methods sacrifice solution optimality for the efficiency and reliability of convex optimization methods.

One may wonder if x_u can be further optimized but, unfortunately, optimizing x_u can be as difficult as x, if not more, because optimizing x_u is non-convex as well and the gradient of x_u is non-trivial to compute since the cost function depends on x_l which further depends on x_u [61]. In [50, 62] finite-difference methods are used to approximate the gradient at the cost of efficiency and precision. Combined with gradient descent, x_u can be optimized using the finite difference approximation of gradients.

At present, not much is known that can be done about the non-convexity of the cost function w.r.t. x_u, but some problems of interest have convex constraints despite having a

non-convex objective. The convexity of constraints makes it easy to maintain feasibility since projection into convex sets is more straightforward than into non-convex sets. For problems with limited computation time budget, solution feasibility is one of the most important requirements in case the optimizer is unable to converge within a limited amount of time. With the constraint convexity assumption, optimization of x_u has feasibility guarantees and is more suitable for real-time applications where early termination may be necessary. As long as a feasible initial guess is provided, algorithms with proper projection such as SNOPT [37] maintain solution feasibility in every iteration. In this thesis, we investigate this type of problem further.

Our strategy is to optimize x_u and x_l hierarchically under a bilevel optimization framework [61, 63]. In the upper level x_u is optimized using gradient descent subject to convex constraints. In the lower level, for given x_u, it uses convex optimization to find x_l, i.e. compute $x_l(x_u)$. The key to efficiently optimize x_u is that the lower optimization can provide analytic gradients of x_u to the upper problem. The analytic gradient helps gradient descent converge more efficiently and reliably than finite-difference approximations [50]. Another reason for the efficiency of the bilevel framework is related to the scaling of variables. Variables x_u and x_l may have vastly different units and scales. The decomposition of x into x_u and x_l allows us to scale them separately. Moreover, the numerical sensitivity of the optimization, i.e. gradients of the cost function with respect to x_u and x_l can be significantly different as well. Solving them together may result in an ill-conditioned problem [64] and separating them may help make both problems better conditioned. This framework is tested on two types of UAV problems in Chapter 2 and 3.

1.2.2 Argmin Local Continuity

According to the theory of parametric optimization [56, 57, 65], the argmin function is locally continuous under conditions of local convexity and constraint qualifications. However, the global behavior of argmin's continuity is unknown but it is reasonable to assume it is at least piecewise continuous for non-degenerate problems of interest. In other words, similar problem parameters usually have similar optimal trajectories. This motivates us to approximate argmin from data collected by the offline optimization of sampled problems.

Assuming the planning task is parameterized by some vector p with known distribution, one may sample $\{p_i\}_{i=1}^n$ from the distribution and solve them offline using trajectory optimization to get a dataset of optimal trajectories $\{z_i\}_{i=1}^n$ where z_i is the optimal trajectory for problem with parameter p_i. The task is use the parameter-solution pairs $\{(p_i, z_i)\}_{i=1}^n$ to build a model $z'(p)$ that approximates the argmin function $z(p)$. This task is called *trajectory*

learning (TL) or *trajectory regression*.

Learning from trajectory data has been investigated for decades. Imitation learning (IL) [66, 67] fits a policy function from data of expert demonstration which usually comes in the form of trajectories. The difference lies in the fact that IL predicts actions directly while TL predicts the trajectory which can be thought as a generalized policy. An issue of IL is the distribution shift that occurs to the states when the policy is applied in a different context than the expert demonstration. Although DAGGER [68] is proposed to address this issue, recollecting expert data may not always be available. Inverse reinforcement learning (IRL) [69] learns a reward function from demonstrations which is then used to optimize the robot's policy. This method is in contrast to our method, which assumes that the objective function is known and uses it to design desired behaviors. This difference also distinguishes our method from learning from demonstration [70] where humans provide direct demonstration to the robots. In TL, the trajectories are computed, rather than empirical, which makes them cheaper and easier to obtain in larger quantities. Learning from precomputed trajectories is also seen in [71] where precomputed plans are used to train a model that predicts the next sampling state for sampling based planners. This approach is usually inefficient for problems with high-dimensional configuration spaces and does not concern multi-modality of the problem.

Trajectory learning can be beneficial in a few ways. A well trained model directly predicts the approximate optimal trajectory, bypassing the computationally expensive numerical optimization. It is also reliable, since it can always compute a feasible trajectory for a given problem. Efficiency is guaranteed as long as the model evaluation is not too complex. The optimality of the prediction is also satisfied as long as the collected data is optimal. The only core assumption of this method is the local continuity of the argmin function, so this framework can be potentially applied in a wide range of applications.

The greatest challenge of TL is to guarantee that it achieves decent accuracy over the whole domain of the argmin function, i.e. it has to perform well over the full support of the distribution of p to guarantee its success rate and avoid the need for refinement by nonlinear optimizers due to the issues in efficiency and robustness of non-convex solvers. One overlooked phenomenon–discontinuity of argmin– hampers the performance of continuous models such as neural networks. Discontinuity of argmin may occur due to a switch of local optimal family and degeneration in some problems, as shown later in this thesis. Neural networks tend to predict an "average" of significantly different trajectories near discontinuity regions [72]. The prediction is far from all the neighboring trajectories and leads to large prediction error and eventually task failure, as shown in numerical experiments later.

It is, therefore, important to properly handle discontinuities in order to guarantee model

performance even in regions near discontinuity. The k-NN model is intrinsically discontinuous and requires no interpolation and can, thus, handle discontinuous functions. However, interpolation is indeed desired since it generalizes better to unknown data especially when data are sparse. It is used in Chapter 4 to provide initial guesses for indirect methods. In this thesis, the major focus is placed on the Mixture-of-Experts (MoE) model. The MoE is composed of a classifier and several experts each of which focuses on a subset of the function domain, all modeled by neural networks. MoE makes predictions by first using the classifier to choose an expert which then predicts the optimal trajectory. It is a discontinuous model since the classifier prediction is discontinuous. As a result, it can potentially avoid the "averaging" issue from continuous models. The details of how to train an MoE are elaborated in Chapter 6. The training procedure requires identification of function discontinuities and splitting of data into pieces within which the argmin function is continuous. For *data splitting* (also called *data clustering*) we propose an empirical approach based on k-Means in Chapter 6 as well as a topology-based method with more theoretical guarantees in Chapter 7. MoE is tested on several benchmark problems and shows advantages over discontinuity-agnostic neural networks in Chapter 6 and 7.

1.3 CONTRIBUTIONS

In summary, this thesis proposes two approaches to exploit two structures within some trajectory optimization problems in order to achieve reliability, efficiency and optimality of the planner. The contributions include:

1. A bilevel optimization framework is proposed for problems with the existence of convex subproblems that is is anytime feasible, efficient, and computes better trajectories than alternatives thanks to the analytic gradient as a by-product of lower level optimization. This framework is tested on the minimum-jerk trajectory problem for UAVs. It shows orders of magnitude improvement over baseline methods and has been tested in physical experiments. This work appeared previously as [73] and is submitted to TRO with co-author Weidong Sun and Kris Hauser. This work is presented in Chapter 2.

2. The bilevel optimization framework is applied to time-optimal problems for UAVs. An approach similar to log barrier on inequality constraints is proposed that helps accelerate the lower level problem and smooth upper level problem. Similarly, its reliability, efficiency and trajectory optimality are better than alternatives that rely on directly solving the full non-convex problem. This work appeared previously as [74],

published in ICRA 2020 with co-author Weidong Sun and Kris Hauser. The details are in Chapter 3.

3. The NNOC framework is proposed that learns from precomputed solutions and predicts good initial guesses for nonlinear optimizers. It is tested on indirect methods where unknown costates have no physical meanings and are difficult to provide initial guesses. It provides significant improvement in terms of the chance of getting globally optimum solutions and computational efficiency compared with random restart techniques. This work appeared previously as [11], published in IROS 2017 with co-author Kris Hauser. The details are in Chapter 4.

4. A learn, predict, and refine framework is proposed that uses neural networks to learn from precomputed problems and predict optimal trajectories, and further local refinement by convex optimization. It shows the capability of TL in complex applications with nonlinear dynamics. The application of this framework on UAVs in physical experiments shows the efficacy of the framework and advantages of obtaining a near-optimal trajectory that considers dynamics over simplified methods that only consider geometry. This work appeared previously as [42], published in IROS 2018 with co-author Weidong Sun and Kris Hauser. This work is presented in Chapter 5.

5. The MoE framework is proposed that is capable of approximating discontinuous function and is better suited to TL where the argmin function is discontinuous. Experiments show the best way to train MoE is to split the discontinuous data into continuous pieces and train an expert for each piece instead of training a single model on the whole dataset through backpropagation. A simple k-Means method with careful tuning of k suffices in many problems and MoE is highly reliable in predicting a trajectory executable by robots, while continuous neural networks tend to fail near discontinuities. This work appeared previously as [75], published in ICRA 2019 with co-author Kris Hauser. The details are in Chapter 6.

6. A topology based framework is proposed that detects if the argmin function to be learned is discontinuous from data, and uses appropriate methods to split the data for better MoE learning. Topological Data Analysis (TDA) computes topological features from data. Any change of topological features is sufficient to prove the discontinuity of the argmin function. Based on the change of topology, the corresponding data split scheme is designed to minimize the number of splits while avoiding discontinuity within each data split. The type of discontinuity from singularity for 3D obstacle

avoidance problem is well handled by the topology-based labeling algorithm while k-Means method struggles with this problem. The details of this work are presented in Chapter 7.

Finally, Chapter 8 concludes this thesis with a summary of the lessons learned from this research and a discussion of potential future works in the field.

CHAPTER 2: FAST UAV TRAJECTORY OPTIMIZATION USING BILEVEL OPTIMIZATION WITH ANALYTICAL GRADIENTS

This chapter proposes a bilevel optimization framework to efficiently solve problems where the convex subproblem structure exists. The problem of time allocation for each piece of a polynomial spline trajectory for UAVs has this structure and is used as an application of this framework. In this problem, the time allocation and the spline coefficients are simultaneously optimized, leading to a challenging non-convex problem. The usual way is to choose time allocations heuristically, and use convex optimization to solve the optimal spline. Our framework improves it by allowing time allocations to be efficiently optimized as well. The key to its efficiency is the analytic gradient is obtained as a by-product of convex optimization which is used to compute the spline coefficients. The bilevel formulation may have non-smooth cost function w.r.t. time allocations and the non-smoothness is handled by subgradient descent with rigorous proof of convergence. This framework has shown orders of magnitude improvement in terms of computational time compared with baselines and has been tested in physical experiments. [1]

[1]This chapter is reproduced from Weidong Sun, Gao Tang and Kris Hauser, "Fast UAV trajectory optimization using bilevel optimization with analytical gradients". In 2020 American Control Conference (ACC). A journal version is under review at IEEE Transactions on Robotics.

2.1 INTRODUCTION

Real-time optimal trajectory generation has long been a challenging but essential component in robotics. Due to the prevalence of nonlinear dynamics, non-convex constraints and high dimensionality of the planning space, it is often difficult to optimize trajectories quickly and reliably. Fortunately for UAVs differential flatness enables trajectory design with polynomials pioneered by Mellinger et al [50]. Polynomial splines are usually used for complex tasks and if the time allocation for each piece of the spline is known,the trajectory can be optimized with Quadratic Programming (QP) methods. This type of method that fixes a subset of optimization variables and optimizes the rest with convex optimization has been applied to path planning for autonomous cars [76, 77], humanoid robots [58] and UAVs [40, 50, 78].

However, this method's optimality relies on the choice of a proper time allocation of the splines when used in UAVs, which, given its intricate and highly nonlinear relationship with the optimization objective and the problem's constraints is not trivial. Indeed, most existing methods rely on heuristics for that task [40] and efficient optimization of time allocation still remains an open question despite efforts in [50, 62]. The lack of ability to efficiently optimize time allocation leads to two situations. In one situation the optimization takes a long time and is thus unable to be used in highly reactive problems. In the other one, heuristics are used and the trajectory has large jerks, which is more likely to fail due to thrust limit and consumes more energy. It is thus necessary to efficiently optimize the time allocation of UAV trajectories.

One approach to this probem was proposed by Mellinger et al. [50], who applied a gradient descent method to refine time allocation. Specifically, they divided the optimization problem in two subproblems (or levels): the lower level optimizes the path while timing is fixed using QP, and the upper level optimizes the time allocation using gradient descent. Similar approach is used in [62].

Nonetheless, finding the gradient of the cost function w.r.t. the time allocation with constraints present in the QP remains an unresolved issue even though analytic gradient is recently given for unconstrained QP [79]. To address this, finite difference method [50] has been employed. But it can be computationally expensive since the number of QPs that needs to be solved at each gradient step grows linearly with the number of spline segments. Moreover, the gradient is inaccurate due to truncation errors and difficulty in the choice of step size. As a result, using finite difference takes longer to converge and tends to converge to a worse cost.

Aiming to address this complexity, we use sensitivity analysis techniques [65] to compute

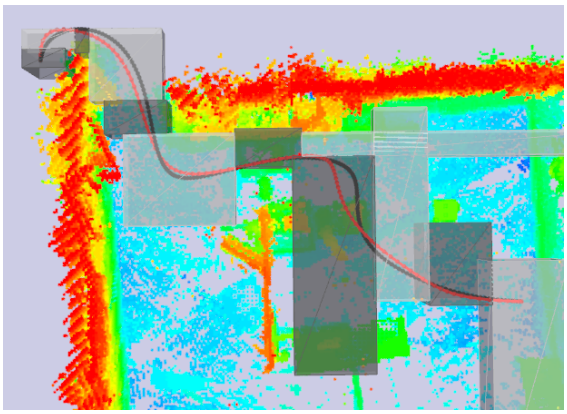
the gradient of the time allocation from the dual solution (Lagrange multipliers) of the QP. By exploiting the dual solution of the QP, our method allows us to compute exact analytical gradients w.r.t. time allocation, bypassing the downsides of finite difference methods: high computational complexity and low accuracy. The gradient of time allocation is used with gradient descent method to optimize the time allocation.

One potential issue is the non-smoothness of the upper level optimization. Fiacco [65] shows that the smoothness of the optimal cost of parametric optimization problems, i.e. objective function of the upper level optimization, requires smoothness of both the cost and constraints, strong convexity near optimum, and some constraint qualifications. In fact, these constraint qualifications do not hold universally and as a result the upper level is theoretically a non-smooth optimization problem. Although this is not a problem in practice, for completeness our method treats objective function discontinuities using a subgradient method when non-differentiability is detected preventing the progress of gradient descent. We prove the convergence of this method despite potential function non-smoothness.

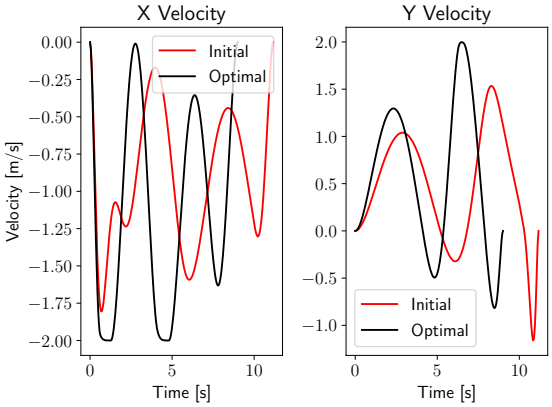
The bilevel optimization framework is guaranteed to yield feasible results at any point in its execution, allowing for arbitrary termination, as long as a feasible initial guess is provided. In contrast, alternative solution methods based on nonlinear optimization such as direct collocation and joint optimization of spatial and temporal variables have no such guarantee due to the problem's strong nonlinearity. Even with feasible initial guesses, our experiments show that these methods do not have a high success rate since these methods do not force feasibility during execution. Compared with straightforward gradient approximation by finite difference, our analytic gradient is more accurate and computationally efficient. As a result, our method outperforms finite-difference baselines in terms of both computation speed and solution quality. One numerical example showing the effectiveness of our technique is in Fig. 2.1. Physical experiments using a real quadrotor on point-to-point navigation and dynamic goal tracking further demonstrate the effectiveness and real-time capabilities of our method.

The contributions of this chapter include

1. A bilevel optimization framework to optimize time allocation with analytic gradient.
2. Comparison with finite-difference, direct collocation, and joint optimization.
3. Theoretical justification of the theorem being used.
4. Subgradient descent method to handle non-smoothness of the cost function and proof of convergence.



(a) Trajectories flying through a point cloud environment with the initial trajectory (red) and optimized trajectory using our method (black). The 12 boxes indicate the safe corridor in which the UAV is constrained.



(b) Velocity profiles of the trajectories in along the x and y direction.

Figure 2.1: Initial trajectory is computed by the heuristic time assignment from Gao et al. [40]. Our method optimizes a weighted sum of total jerk and time. Note for this one a large weight is placed on total time and the trajectory is aggressive since it reaches velocity limits for a large portion of time.

5. Test of the algorithm in complex environments and realistic environments represented by point clouds.

6. Theoretical and empirical scalability analysis to problems with more than 40 segments.

7. Demonstration of real-time capability in physical experiments.

2.2 RELATED WORK

2.2.1 Trajectory Optimization for UAVs

Trajectory optimization solves the problem of computing the optimal trajectory for dynamic systems under some cost function and constraints. It is a widely used method for motion planning of robotic systems. See [80] for an introduction of trajectory optimization for general systems. Specifically, direct collocation [81] has been widely used. However, pioneered by Mellinger [50], polynomial trajectories which exploit the differential flatness of UAV dynamics are often used for UAVs. With the snap of the trajectory as the cost function, a quadratic optimization suffices to compute the optimal trajectory. Some extensions to this framework include using Bézier curve control points as optimization variables and

safe corridor generation to guarantee collision avoidance [40]. However, to use this framework, the temporal variables of the trajectory such as the duration of each segment have to be chosen prior to optimization. They are usually chosen by heuristics and this provides room for optimization of time allocations. Wang et al. [82] propose an alternating method but it lacks the ability to handle complex spatial constraints. Recent work [83] uses mixed-integer QP to solve for trajectories and guarantees safety by always having a feasible, safe back-up trajectory. However, the mixed-integer QP does not scale well with the number of convex polyhedra. Our paper demonstrates the scalability of our method, which can solve problems in corridors with up to more than 40 polyhedra. Gao et al. [41] computes safe and aggressive trajectories in real time from human-piloted trajectories, and generates them by alternatively optimizing the spatial and temporal trajectories. However, the two parts optimize different objective functions, so convergence is not guaranteed. On the contrary, our bilevel optimization method has been proven to converge to a local minimum of a unified objective.

2.2.2 Optimizing Time Allocation

As described above, trajectory optimization with polynomial splines is a well-studied problem as long as the spline timing is fixed. However, finding an optimal time allocation in real-time is still challenging. One strategy [40, 78] is to use heuristics such as graph search on a discretized grid to generate a time allocation and keep timing fixed during the optimization stage. Iterative methods such as gradient descent [50, 62] have also been used to optimize time allocation. However, if gradients are computed by finite differences, $(n + 1)$ QPs have to be solved for a problem with n segments for every gradient evaluation, making it slow and inaccurate. Another strategy to determine time allocation is to use sampling [58]. This approach randomly samples the duration of each spline segment until the corresponding QP can be solved, and has been applied successfully to humanoid locomotion problems. Because optimality is not emphasized, this method is only helpful in settings where obtaining a feasible solution is the major bottleneck.

2.2.3 Bilevel Optimization

Bilevel optimization [63] refers to a mathematical program where one optimization problem (the upper-level optimization problem) has another optimization problem (the lower-level optimization problem) as one of its constraints, i.e., one optimization task is embedded within another.

Bilevel and multi-level optimization techniques have been employed for switching time optimization for switched systems [84, 85, 86, 87]. These works focus on calculating derivatives of an objective function with respect to switching times. In particular, works by Xu et al. [84] and Egerstedt et al. [85] compute the derivatives using Lagrange multiplier methods, which bear some resemblance to sensitivity analysis technique used in this paper. However, these works are based on Pontryagin's Maximum Principle [80] and fall short of the capability to include inequality path constraints, which is often unavoidable in robotic applications.

Applications of bilevel optimization in robotics include trajectory optimization for legged robots [87], and robust control and parameter estimation [88]. Landry et al. [88] present a bilevel optimization solver based on an augmented Lagrangian method, however their bilevel method is slower than directly solving the NLP in their experiments.

Many algorithms are available to solve bilevel optimization, and we refer readers to Sinha et al. [63] and Colson et al. [61] for more comprehensive treatments of the topic. Most closely related to our approach is the *descent method*, which seeks to decrease the upper-level objective while keeping the new point feasible. Our method might be categorized as a descent method as we solve the upper-level optimization problem by gradient descent using gradients provided by the lower-level optimization problem.

2.2.4 Gradients in Bilevel Optimization

Efficiently and accurately computing gradients of the lower-level optimization problem is essential in applying gradient-based methods to solve bilevel optimization problems. The key derivation used in our algorithm is based on sensitivity analysis for parametric NLPs [65]. The method used to compute gradient in this work is very similar to Pirnay et al. [89], which provides the optimal sensitivity of solutions to NLP problems.

Efficient computation of gradients of optimization problems is also explored in the field of machine learning. OptNet [90] incorporates a QP solver as a layer into the neural network and is able to provide analytical gradients of the solution to the QP with respect to input parameters for back propagation. Gould et al. [91] presents results on differentiating argmin optimization problems with respect to optimization variables in the context of bilevel optimization.

2.3 METHODOLOGY

In this section, we describe our framework. We start by a mathematical formulation of the trajectory optimization problem for UAVs and end with a description of our algorithm.

2.3.1 Trajectory Optimization Preliminaries

In the case of UAV motion planning, differential flatness allows us to plan a trajectory in the UAV's four flat outputs $[p(t)^T\ \psi(t)]^T$ which consist of 3-D position $p(t) \in \mathbb{R}^3$ and yaw angle $\psi(t) \in SO(2)$, without explicitly enforcing dynamics [50]. In this work, we plan in $\mathbb{R}^3$ by assuming the yaw stays constant, which is a common practice in UAV motion planning.

Trajectory optimization aims to find a trajectory $x : [0, T] \to \mathbb{R}^d$ that minimizes some measure of performance J while satisfying all the necessary constraints, e.g. being collision-free and dynamically feasible, and is formulated as:

$$\begin{aligned}
\underset{x,T}{\text{minimize}} \quad & J(x,T) = \int_0^T \ell(x,t)\mathrm{d}t + \Phi(x(T)) \\
\text{subject to} \quad & x(0) = x_0 \\
& x(T) \in \mathcal{X}_{\text{goal}} \\
& \phi(x(t)) \leq 0, \quad \forall t \in [0,T] \\
& \theta(x(t)) = 0, \quad \forall t \in [0,T],
\end{aligned}$$

which is similar to the definition in Chapter 1 Eq. (2.3) except that the state vector x now only contains the 3-D position $p(t)$; constraints of dynamical equation are neglected due to differential flatness; and constraints are split into equality and inequality constraints ϕ and θ, respectively. Collision avoidance is the major constraint and encoded in g. Other constraint include limits of velocity and acceleration.

2.3.2 Safe Corridors

To guarantee that the whole trajectory will stay collision-free, we extract a safe corridor from the environment using the implementation from Gao et al. [40]. Here we give a brief overview of their method. Given a map represented by an occupancy grid or an Euclidean signed distance field (ESDF), a start position and a goal position, a safe corridor is generated by taking the following steps:

1. Inflate all the obstacles in the map by a safety radius, so that the UAV can be considered as a point.

2. Find a path that connects the start and the goal using $A^\star$ search or fast marching method (FMM).

3. Grow a safe corridor consisting of convex polytopes around the path. In our implementation, we generate axis-aligned boxes by growing an axis-aligned box centered around

each node in the path until it hits an obstacle, and then removing redundant boxes.

One such corridor and its corresponding environment [2] is shown in Fig. 2.1a. Here, axis-aligned boxes are chosen for simplicity and compatibility with the grid data structures commonly used by perception algorithms. But in general, our method is applicable to corridors composed of any convex polyhedron.

2.3.3 Trajectory Optimization with Piecewise Bézier Curves

In this section we define a trajectory optimization problem in terms of spatial variables c (polynomial coefficients) and temporal variables y (durations of each segment of the curve).

We represent the trajectory as a piecewise Bézier curve of order d with n segments and segment durations $\Delta t_1, \ldots, \Delta t_n$. The timing of each knot point (connection point between two consecutive pieces) is given by $t_i = t_{i-1} + \Delta t_i$ with $t_0 = 0$. The i'th segment is defined over the domain $[t_{i-1}, t_i]$ as:

$$x(t) = \sum_{j=0}^{d} c_{ij} B_{d,j}\left(\frac{t - t_{i-1}}{\Delta t_i}\right), \quad t \in [t_{i-1}, t_i]$$

for each $i = 1, \ldots, n$, where $c_{ij} \in \mathbb{R}^3$ denotes the j'th control point in the i'th segment and $B_{d,j}$ denotes the j'th Bernstein polynomial of order d defined as

$$B_{d,j}(u) = \frac{d!}{j!(d-j)!} u^j (1-u)^{d-j}.$$

We gather all the polynomial coefficients (control points) in the flattened vector $c \in \mathbb{R}^{3n(d+1)}$ and define the time allocation as $y = [\Delta t_1, \ldots, \Delta t_n]^T \in \mathbb{R}^n_+$.

Objective Function Often, the objective function J is chosen to be the integral of the squared norm of some high-order derivative of the trajectory to penalize control effort. We use a more general objective:

$$J(x, T) = \int_0^T \|x^{(q)}(t)\|^2 dt + wT, \tag{2.1}$$

which is a weighted sum of the integral of the squared L_2-norm of the q'th derivative and the traversal time T, with weighting parameter denoted by w.

[2] `http://ais.informatik.uni-freiburg.de/projects/datasets/octomap/`. Last retrived Jul-31-2020.

It has been shown that the first term in Eq. (2.1) can be written as a quadratic function of the coefficients c, with the quadratic matrix $P_q(y)$ determined by time allocation y and the order of derivative q [40, 50]. With a slight abuse of notation, we can write Eq.(2.1) in terms of polynomial coefficients c and time allocation y:

$$J(c, y) = c^T P_q(y) c + \mathbb{1}^T y, \tag{2.2}$$

where $P_q(y)$ is a symmetric positive semidefinite matrix that is nonlinear in y and $\mathbb{1}$ is a vector of 1s. Note that we will drop the subscript q in $P_q(y)$ from now on since it is assumed to be $q = 3$ (we wish to find a minimum jerk trajectory) throughout this work.

Constraints on Continuity Constraints on the trajectory should be enforced so that:

a) States at the start and end of the trajectory should match the initial state and (optional) final state.

b) Continuities at knot points which ensure a smooth transition between each segment of the trajectory. If the trajectory needs to be C^k continuous, equality constraints up to the k'th order should be applied at all knot points. We found that in the UAV case, applying continuity constraints up to acceleration yields good results, the same as [40].

The above constraints can be compiled into a linear equality constraint on the polynomial coefficients c:

$$H(y)c = h, \tag{2.3}$$

where the matrix H is generally nonlinear in time allocation y.

Constraints on Safety and Dynamic Feasibility Safety and dynamical feasibility are ensured by imposing inequality constraints such that:

1. The whole trajectory stays in the safe corridor discussed in Section. 2.3.2.

2. The maximum velocity $\|x'(t)\|_\infty$ and maximum acceleration $\|x''(t)\|_\infty$ along the trajectory are bounded, i.e.,

$$\|x'(t)\|_\infty \leq v_{\max}, \quad \|x''(t)\|_\infty \leq a_{\max} \quad \forall t \in [0, T] \tag{2.4}$$

with $v_{\max}$ and $a_{\max}$ prescribed by the capabilities of the vehicle, user preference or operational norms.

We encode the trajectory using a piecewise Bézier curve [40], which has the properties:

1. The curve is totally contained in the convex hull of its control points.
2. The derivative of a Bézier curve is again a Bézier curve, with its coefficients being a linear combination of its antiderivative's coefficients.

Using these properties safety and dynamically feasible constraints can be imposed as a linear inequality constraint on the flattened coefficients [40]:

$$G(y)c \leq g, \tag{2.5}$$

where matrix G is generally nonlinear in time allocation y.

We note that constraining position, velocity and acceleration using control points of Bézier curve does introduce some conservativeness since only the ends of the curve reach the convex boundary even if all control points are at boundary. This effect can be seen in Fig. 2.1b, in which the velocity limits of $\pm 2\,\mathrm{m/s}$ are only reached at a few discrete points.

An alternative collocation implementation would use a grid along the trajectory, with constraints applied at grid points. Although collocation is less conservative, it would not guarantee feasibility at non-grid points. Another approach is to split the Bézier curve into more pieces so the curve can be represented by more control points. We refer readers to [51] for a more detailed discussion about the conservativeness of Bézier curve in a convex hull and potential alternatives.

If the total trajectory time can be changed, one can optimize without velocity and acceleration bounds and simply increase the total trajectory time to satisfy the bounds without losing optimality of the time allocation. In fact, since our objective minimizes jerk, the dynamic feasibility is already considered in the cost function to some extent. No matter what representation is used, only the lower level optimization is affected and the efficacy of our analytic gradient computation and gradient descent method still holds.

Constraints on Time Hard constraints on time allocation y may be imposed, such as a fixed total traversal time or that the duration of each segment must be positive. We encode these constraints as

$$Ay \leq b, \quad Cy = d, \tag{2.6}$$

with A, b, C, d properly chosen.

2.3.4 Final Formulation

In summary, we collect Eq. (2.2), (2.3), (2.5) and (2.6), into the problem of Trajectory Optimization using Bézier spline in a Corridor (TOBC):

$$\begin{aligned} \underset{c,y}{\text{minimize}} \quad & J(c,y) = c^T P(y) c + w \mathbb{1}^T y \\ \text{subject to} \quad & Ay \leq b \\ & Cy = d \\ & G(y)c \leq g \\ & H(y)c = h, \end{aligned} \tag{TOBC}$$

which is nonlinear in time allocation y and convex (quadratic) in spline coefficients c for fixed y. This formulation generalizes the formulations found in [40, 58, 77, 78].

We mainly study two variants of (TOBC). The Hard Time variant imposes a fixed traversal time and uses minimum-jerk as the objective (following Mellinger and Kumar [50]):

$$\begin{aligned} \underset{c,y}{\text{minimize}} \quad & J(c,y) = c^T P(y) c \\ & \mathbb{1}^T y = T_c \\ & y \geq \delta \\ & G(y)c \leq g \\ & H(y)c = h, \end{aligned} \tag{2.7}$$

where T_c is a fixed traversal time, e.g., chosen by a higher-level planner, and δ is a small value (we use 1×10^{-6}) which ensures that durations are positive in each corridor. We note that the fixed traversal time T_c is necessary.

The Soft Time variant uses a weighted sum of jerk and traversal time as the objective (following Richter et al. [62]) rather than imposing hard constraints on traversal time:

$$\begin{aligned} \underset{c,y}{\text{minimize}} \quad & J(c,y) = c^T P(y) c + w \mathbb{1}^T y \\ & y \geq \delta \\ & G(y)c \leq g \\ & H(y)c = h. \end{aligned} \tag{2.8}$$

One use case of Eq. (2.8) is tracking a dynamic goal, with the tracking aggressiveness tuned by the weight w.

2.3.5 Formulation of the Bilevel Optimization Problem

To efficiently solve (TOBC), we will rewrite it as a bilevel optimization problem, which is defined as follows [63].

Definition 2.1. A bilevel optimization problem is given by

$$\begin{aligned}
\underset{x_u \in X_U, x_l \in X_L}{\text{minimize}} \quad & F(x_u, x_l) \\
\text{subject to} \quad & x_l \in \underset{x_l \in X_L}{\operatorname{argmin}} \{ \quad f_0(x_u, x_l) : \\
& \qquad\qquad\qquad g_i(x_u, x_l) \leq 0, i = 1, \ldots, m \\
& \qquad\qquad\qquad h_i(x_u, x_l) = 0, i = 1, \ldots, p\} \\
& G_i(x_u, x_l) \leq 0, \quad i = 1, \ldots, M \\
& H_i(x_u, x_l) = 0, \quad i = 1, \ldots, P
\end{aligned}$$

where the upper-level optimization problem is defined by upper-level objective $F(\cdot)$ and upper-level constraints encoded in $G(\cdot)$ and $H(\cdot)$. The lower-level optimization problem, defined by lower-level objective f_0, with lower-level constraints $\{f_i(\cdot)\}_{i=1}^m$ and $\{h_i(\cdot)\}_{i=1}^p$, is embedded as a constraint in the upper-level optimization problem. The upper-level and lower-level decision variables are x_u and x_l, respectively. In Fig. 2.2 we illustrate a simple bilevel optimization problem where $x_u \equiv y$ encodes the constraints and x_l is the lower-level variable to be solved for every instance of y. In this problem for every y the constraint is different, and so is the optimal solution. In general, both the cost function and constraint may depend on the upper level variable x_u but here only the constraint depends on x_u. The goal in this problem is to find the optimal y such that the corresponding lower-level problem has the smallest cost. Our analytic gradient computes the gradient of the lower level optimal cost with respect to the upper level variable x_u.

However, the bilevel optimization problem defined in Definition 2.1 is in general difficult to solve if no structure exists such as convexity of the lower-level problem or the lower-problem has closed-form solution [61]. Luckily, in our problem one structure is the lower and upper problems share the same objective function, and the original problem can be written as

$$\begin{aligned}
\underset{x_u \in X_U, x_l \in X_L}{\text{minimize}} \quad & F(x_u, x_l) \\
\text{subject to} \quad & G_i(x_u, x_l) \leq 0, \quad i = 1, \ldots, M \\
& H_i(x_u, x_l) = 0, \quad i = 1, \ldots, P
\end{aligned} \tag{2.9}$$

which can be solved by *joint optimization* of x_l and x_u by nonlinear optimizer. The bilevel

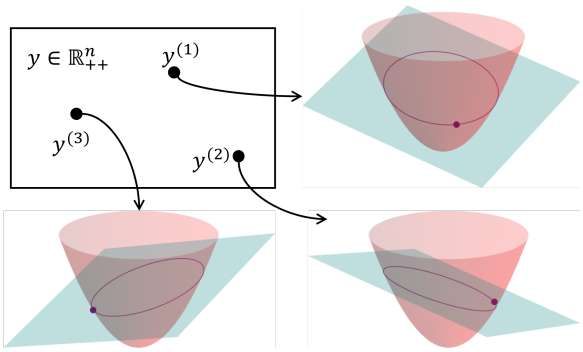


Figure 2.2: An illustration of bilevel optimization: $y^{(1)}$, $y^{(2)}$ and $y^{(3)}$ are three feasible time allocations, they are optimized in the upper-level optimization problem. Each of these y corresponds to a quadratic programming problem, which is being solved in the lower-level optimization problem. The red paraboloids are quadratic objective functions, cyan planes are equality constraints, no inequality constraints are drawn for illustration purposes. Feasible sets are purple curves, and purple dots are the optimal solutions to each lower-level optimization problem.

formulation is

$$\begin{aligned}
\underset{x_u \in X_U, x_l \in X_L}{\text{minimize}} \quad & F(x_u, x_l) \\
\text{subject to} \quad & x_l \in \underset{x_l \in X_L}{\text{argmin}}\{ \quad F(x_u, x_l) : \\
& \qquad\qquad\qquad\quad G_i(x_u, x_l) \le 0, i = 1, \ldots, M \\
& \qquad\qquad\qquad\quad H_i(x_u, x_l) = 0, i = 1, \ldots, P\} \\
& G_i(x_u, x_l) \le 0, \quad i = 1, \ldots, M \\
& H_i(x_u, x_l) = 0, \quad i = 1, \ldots, P
\end{aligned} \tag{2.10}$$

Note that the only difference is the bilevel formulation have additional constraints that are automatically satisfied at the optimum of the original problem in Eq. (2.9). To simplify Eq. (2.10) the constraints in $\{G_i\}_{i=1}^M$ and $\{H_i\}_{i=1}^P$ are split into two groups–one group (potentially empty) depends on x_u only and the other one contains the remaining constraints. For simplicity, they are written as $\varphi_u(x_u) \le 0$ and $\varphi_l(x_u, x_l) \le 0$ with appropriate dimensions. Note equality constraints are expressed using two inequality constraints in φ_u and φ_l. The problem is then written as

$$\begin{aligned}
\underset{x_u \in X_U, x_l \in X_L}{\text{minimize}} \quad & F(x_u, x_l) \\
\text{subject to} \quad & x_l \in \underset{x_l \in X_L}{\text{argmin}}\{ \quad F(x_u, x_l) : \\
& \qquad\qquad\qquad\quad \varphi_l(x_u, x_l) \le 0\} \\
& \varphi_u(x_u) \le 0
\end{aligned} \tag{2.11}$$

The lower problem allows to write its optimum x_l^* as a function of x_u (assuming the lower problem has unique solution so we can call it function instead of set mapping, but the framework also works if the solution is indeed a set) and the upper level problem is now

$$\begin{array}{ll} \underset{x_u \in X_U}{\text{minimize}} & F(x_u, x_l^*(x_u)) \\ \text{subject to} & \varphi_u(x_u) \leq 0 \end{array} \tag{2.12}$$

where the objective function only depends on x_u (potentially non-convex and non-smooth) and constraints of x_u are convex. This formulation assumes the minimizer of the lower problem x_l^* exists and is unique for every $x_u \in X_U$. While this is in general not true, but for the problem studied in this chapter, the uniqueness of the lower problem can be proved. The existence, however, is seldom an issue since infeasible lower problem means the objective of the upper problem is infinity and should be avoided in upper level optimization.

However, the upper level optimization problem is usually difficult to solve for general bilevel problem, and sometimes more challenging than solving the original problem without decomposition even if the lower problem is convex. In the upper problem function evaluation becomes more expensive since the lower problem has to be solved for every evaluation. Even if the lower problem is convex and generally efficient to solve, it is still much more computationally expensive than simply evaluating the objective and cost functions in the original problem. Moreover, one has to differentiate $F(x_u, x_l^*(x_u))$ to compute the gradient w.r.t. x_u in order to optimize it unless derivative-free methods are used which tend to be inefficient. The differentiation is highly non-trivial since it requires to differentiate the optimizer of the lower problem which may not be expressed in closed-form. In [50, 62] finite-difference is used to approximate the gradient at the cost of efficiency and precision. Moreover, the optimizer of the lower problem may be non-smooth w.r.t. x_u if some constraint qualification are not satisfied [65], leading to non-smooth upper cost function. The upper problem may be constrained if ϕ_u is not empty which may impose feasibility challenges to the upper problem. As a result, the upper problem may be more challenging than the original problem.

Following definition in Eq. (2.11), we rewrite (TOBC) as:

$$\begin{array}{ll} \underset{c,y}{\text{minimize}} & J(c,y) = c^T P(y) c + w \mathbb{1}^T y \\ \text{subject to} & c \in \underset{c}{\text{argmin}} \{ J(c,y) : G(y)c \leq g,\ H(y)c = h \} \\ & Ay \leq b \\ & Cy = d. \end{array} \tag{TOBC-BO}$$

Note that although the objective functions $J(c, y)$ remain the same in both the lower-level (also the first constraint) and upper-level optimization problem, y is fixed in the lower-level optimization problem but becomes the optimization variable in the upper-level optimization problem. The lower-level problem is also a quadratic program (QP) because $J(c, y)$ is quadratic in c when y is fixed.

Our solution strategy is to use constrained gradient descent on the function $J^\star(y) = J(c^\star(y), y)$ with $c^\star$ minimizing the QP for every time allocation y, i.e.,

$$c^\star(y) \in \underset{c}{\operatorname{argmin}}\{J(c, y) : G(y)c \leq g,\ H(y)c = h\}. \tag{2.13}$$

Gradient descent has, indeed, been used to solve bilevel optimization problems [63], and our framework is a variant of this method. Note that in Eq. (TOBC-BO) and (2.13) we express $c \in \operatorname{argmin}$ since the solution to the lower problem is in general a set. However, we prove in Sec. 2.4 that the optimal solution to TOBC is unique. Given a feasible $y \in \mathbb{R}^n$, we find a direction $-\nabla_y J^\star(y) \in \mathbb{R}^n$ and a step length α that can make a sufficient decrease in $J^\star(y)$ while maintaining the feasibility of the new point $y_{\text{new}} = y - \alpha\nabla_y J^\star(y)$. The key issue with this approach is obtaining the gradients, which we address in the next section.

For this problem, a bilevel optimization framework may indeed outperforms joint optimization for several practical reasons. The lower level solves spatial variables and utilizes off-the-shelf convex solvers to handle the strong sensitivity of polynomial coefficients and constraints for collision avoidance. The upper level optimizes temporal variables that are numerically better behaved with simple linear constraints after being decoupled with spatial variables. using gradient descent with backtracking line search [64] and gradient projection.

Because gradient descent does require a feasible initial guess of y, it is worth describing how such a feasible point can be determined. For TOBC, feasibility requires meeting velocity and acceleration limits, which requires enough time to be allocated to segments. If the initial and final states are static (with zero velocity and acceleration), one can increase the total trajectory time by scaling the time allocation to make it feasible. In other cases, however, an initial guess may not be feasible. One possible approach to mitigate this problem is to define a relaxed inner problem with slack variables if the initial guess is infeasible. We leave this problem to be addressed in future work.

2.3.6 Gradient Computation

We use a key result from sensitivity analysis of parametric nonlinear programming (NLP) [92, Thm 2.3.3] to derive the gradient $\nabla J^\star(y)$. In our problem, the lower-level objective is the

same as the upper-level one, which allows us to derive gradients of the upper-level decision variables through sensitivity analysis. For brevity, we give results of first-order sensitivity analysis and refer readers to [92] for details.

Theorem 2.1. Consider the problem of finding the local solution $c(y)$ of a parametric NLP problem:

$$\begin{aligned} \underset{c}{\text{minimize}} \quad & J(c, y) \\ \text{subject to} \quad & g_i(c, y) \leq 0, \quad i = 1, \ldots, m \\ & h_j(c, y) = 0, \quad j = 1, \ldots, p \end{aligned}$$

where c is the vector of decision variables and $y \in \mathbb{R}^n$ is a parameter vector. Let $c^\star(y_0)$ be a locally optimal solution, and let λ and ν be Lagrange multipliers associated with $g(\cdot)$ and $h(\cdot)$, respectively.

If the following conditions hold:

1. functions $J(\cdot)$, $g_i(\cdot)$ (for all i) and $h_j(\cdot)$ (for all j) are twice continuously differentiable in c, and their gradients w.r.t. c and the constraints $g_i(\cdot)$ (for all i) and $h_j(\cdot)$ (for all j) are once continuously differentiable in y in a neighborhood of $(c^\star, y_0)$,
2. objective $J(c, y)$ is twice continuously differentiable in (c, y) near $(c^\star, y_0)$,
3. the strong second-order sufficient conditions (SSOSC) hold at $c^\star(y_0)$,
4. the gradients $\nabla g_i(c^\star, y_0)$ (for i such that $g_i(c^\star, y_0) = 0$) and $\nabla h_j(c^\star, y_0)$ (for all j) are linearly independent,

then in a neighborhood of $y = y_0$, the gradient of the objective is

$$\nabla_y J^\star(y) = \nabla_y J + \sum_{i=1}^{m} \lambda_i(y) \nabla_y g_i + \sum_{j=1}^{p} \nu_j(y) \nabla_y h_j. \tag{2.14}$$

Moreover, similar theorem in [65] further requires the Strict Complementary Slackness (SCS), i.e., $\lambda_i > 0$ when $g_i(c^\star, y_0) = 0$ but only requires second-order sufficient condition. With SCS, the active set which is the collection of inequality constraints where equality holds i.e. $\{i \in \{1, \ldots, m\} | g_i(c^*, y_0) = 0\}$ does not change. Theorem in [92] does not require SCS and this means $J^*(y)$ is differentiable even if some constraints switches between being active and non-active.

In our problem, conditions 1) and 2) are satisfied by construction; SSOSC can be proved (see Appendix 2.7.1 for details). However, condition 4), also known as the Linear Independence Constraint Qualification (LICQ) may fail in some cases. Moreover, this condition

can only be verified after the NLP is solved and the pattern of active constraints is known. When LICQ holds, the Lagrangian multipliers are unique and Eq. (2.14) computes the exact gradient of the objective. When LICQ does not hold, however, the Lagrangian multipliers associated with linearly dependent constraints are not unique. The Karush-Kuhn-Tucker (KKT) condition is satisfied for infinite number of multipliers. As a result, the gradient computed by Eq. (2.14) may not be unique. In Sec. 2.4.3, we show that under the assumption of Slater's condition [93], Eq. (2.14) computes a subdifferential. Slater's condition only requires the existence of a feasible solution in the interior of the convex feasible set and is not a strict assumption.

2.3.7 Solving Bilevel Optimization

Our algorithm, given in Alg. 2.1, uses a combination of gradient descent and subgradient descent to solve (TOBC-BO). It takes an initial guess of the time allocation y_0 as input. It then iteratively descends $J^\star(y)$ until some optimality conditions are satisfied or the maximum number of iterations is reached. The usual step is a gradient descent step, but if it is detected that the function is nondifferentiable, the algorithm switches to take a subgradient step. Subgradient descent is widely used in training deep neural networks for non-convex and non-smooth objective functions. Subgradient descent alone, however, is quite slow due to diminishing step sizes. In our problem, the function is smooth in most regions so gradient descent with line search is usually far more efficient. As we shall see in the experiments, the subgradient step is rarely taken, but can kick the algorithm out of states where gradient descent gets stuck.

Line 3 solves a QP problem with a time allocation y and then returns the objective value J and the dual solution (Lagrange multipliers) λ and ν. Line 4 computes the estimated gradient of the objective w.r.t. time allocation y with the Lagrange multipliers λ and ν using Eq.(2.14). Line 5 finds a normalized descent direction from the gradient by projecting the gradient onto the null space of equality constraints $Cy = d$ [64]. In the Hard-Time variant that we consider, the constraint is that total time is a constant, and hence the projected gradient is computed as $p = g - \frac{1}{n}\sum i = 1^n g_i$.

Line 6 calls the line search Alg. 2.2 to find a suitable step length α that meets the inequality constraints on y and gives sufficient decrease in the objective function. If α cannot be found, we assume y is near a non-smooth region, so Alg. 1 takes a subgradient step without checking for sufficient decrease. If taken, the first subgradient step size α_{sub} is initially set to the initial α at the step when line search fails. If a line search step can be found, the following optimality conditions are checked in Line 7:

Algorithm 2.1: Refine-Time $(y_0, \alpha_{\text{sub}})$

```
1  y ← y_0, n_sub ← 0, J_opt ← ∞, y_opt ← 0
2  for i ← 0 to max-iterations do
3  |   J, λ, ν ← Solve-QP(P(y), G(y), g, H(y), h)
4  |   g ← Get-Gradient(λ, ν) // From Eq. (2.14)
5  |   p ← Project-Gradient(g, A, b, C, d)
6  |   α, J, y ← Line-Search(y, p)
7  |   if α not found then
8  |   |   y ← y − α_sub p/(n_sub + 1), n_sub ← n_sub + 1
9  |   |   J ← J(y)
10 |   else
11 |   |   if optimality-conditions-satisfied then
12 |   |   |   break
13 |   |   end
14 |   end
15 |   if J < J_opt then
16 |   |   (J_opt, y_opt) ← (J, y)
17 |   end
18 end
19 Return y_opt
```

1. Norm of the projected gradient is less than 1×10^{-3}.

2. The change of the absolute or relative objective function is less than 1×10^{-3}.

Since the subgradient step may actually increase the objective function's value, the best solution during all iterations is returned (Lines 12–14).

We use an adaptive backtracking method to find a step α to achieve the Armijo sufficient decrease condition [64]. If it fails to find a decrease, "α not found" will be returned as in Line 16. The initial step length α_0 defined in Line 1 will be updated adaptively. The update strategy is similar to the update of trust region radius in a trust region algorithm [64]: α_0 will grow or shrink based on the decrease achieved in the first iteration, shown in Line 8 and Line 10, respectively. We find adaptive line search useful since it reduces the number of QPs that are solved during line search and time for solving QPs dominates our algorithm's time complexity.

For a candidate step $y \to y - \alpha p$ where α is the step size and p is the gradient, a non-smoothness occurs when some point along this line segment fails to meet the LICQ condition and first-order Taylor expansion does not well approximate the function value. However, we do not attempt to detect every point at which the objective is non-smooth, because the gradient can still make adequate progress (as determined by the sufficient decrease

Algorithm 2.2: Line-Search (y_s, p)

static variable α_0 **constant variables** $\tau_g > 1, 1 > \tau_s > 0$
$\alpha \leftarrow \alpha_0$
for $i \leftarrow 0$ *to max-iterations* **do**
    $y = y_s - \alpha p$
    $J, \lambda, \nu \leftarrow$ Solve-QP$(P(y), G(y), g, H(y), h)$
    if *sufficient-decrease-achieved* **then**
        if $i = 0$ **then**
            $\alpha_0 \leftarrow \tau_g \alpha$
        else
            $\alpha_0 \leftarrow \tau_s \alpha$
        end
        Return α, J, y
    end
    $\alpha \leftarrow \tau_s \alpha$
end
Return "not found", J, y

condition) and it is practically computationally expensive to do. Instead, we decide to trigger the subgradient step only when backtracking line search fails to find a sufficient cost decrease. The subgradient steps use a standard diminishing step size to guarantee convergence. We justify this decision further with the convergence analysis of Sec. 2.4. Moreover, our experiments in Sec. 2.5.3 suggest that introducing subgradient steps increases the number of iterations, but with the benefit of providing more opportunities to improve the objective.

2.4 CONVERGENCE ANALYSIS

Although gradient descent with line search is convergent in smooth regions [64], we must study the behavior of Alg. 2.1 in non-smooth regions when it switches to subgradient methods. The proof is mainly based on results from Davis et al. [94] which prove the convergence of the stochastic subgradient descent method for a wide variety of functions including those represented by deep neural networks. We prove convergence by showing the cost function of the upper optimization in our problem satisfies the conditions of the theorem in [94]. We note that the proof relies on the very structure of our problem and is not necessarily true for other bilevel optimization problems. Specifically, the main property we use is the convexity of the lower level problem.

2.4.1 Clarke Subdifferential

For a locally Lipschitz continuous function $f : \mathbb{R}^d \to \mathbb{R}$, the *Clarke subdifferential* [95, Ch2, Theorem 8.1] of f at any point x is the set

$$\partial f(x) \equiv \text{conv}\left\{\lim_{i\to\infty} \nabla f(x_i) : x_i \xrightarrow{\Omega} x\right\} \tag{2.15}$$

where Ω is any full-measure subset of $\mathbb{R}^d$ such that f is differentiable at each of its points; conv means the convex hull of the limit of gradient for all sequences in Ω approaching x and a point is *(Clarke) critical* if $0 \in \partial f(x)$. An arc $x(t) : \mathbb{R}_+ \to \mathbb{R}^d$ is called a *trajectory* if it satisfies the differential inclusion

$$\dot{x}(t) \in -\partial f(x(t)) \quad \text{for a.e. } t \geq 0 \tag{2.16}$$

An iteration sequence is used to track the trajectory

$$x_{k+1} = x_k + \alpha_k(g_k + \xi_k) \tag{2.17}$$

where $\alpha_k > 0$ is a sequence of step sizes that is square summable but not summable, $g_k \in -\partial f(x)$, and ξ_k is noise which is zero in our case. Then by [94, Theorem 3.1] the sequence generated by Eq. (2.17) approximates the trajectory of the differential inclusion.

2.4.2 Convergence Result

In order for the trajectory of differential inclusion to converge to a critical point, a sufficient condition is that $f(x)$ must locally Lipschitz and semianalytic [94, Theorem 5.9]. This condition requires $f(x)$ to be piecewise analytic which fits naturally in our case where different combinations of active inequality constraints result in piecewise functions. It suffices to show within each piece, that the optimal cost of the lower optimization is an analytic function. Considering that the lower problem is a QP which can be solved by inverting a matrix whose entries are analytic functions of the time allocation y given $y > 0$, the optimal solution is an analytic function, and, consequently, so is the cost function of lower optimization.

It remains to show that each piece is connected, i.e. the optimal lower cost is continuous with respect to the time allocation y. This can be shown by Theorem 2.1 of [65] which states that the optimal cost is continuous as long as: the set-valued mapping from time allocation to the feasible set (of lower optimization) is continuous; the feasible set is compact; and the objective function is continuous. These conditions are easy to verify in our problem. As a

result, the subgradient methods being used in Alg. 2.1 converges to a critical point as long as 1) the gradient from Eq. (2.14) is indeed a subdifferential even when LICQ fails to hold and 2) the constraints in the upper level optimization do not affect the convergence of the algorithm. These two conditions are discussed in the next two sections.

2.4.3 LICQ Failure

LICQ fails when, at the optimum point, the linearized equality and active inequality constraints are linearly dependent. In that case, the Lagrangian multipliers associated with those constraints are non-unique. In our problem, the lower optimization is QP and, thus, if LICQ fails to hold, it means that some rows of H and $G_{\mathcal{A}}$ are linearly dependent where $G_{\mathcal{A}}$ is the collection of inequality constraints that are active. Commercial QP solvers like Sqopt and Mosek have a pre-solve process that eliminates redundant constraints so the actual QP being solved returns unique multipliers and the redundant constraints have multipliers of zero.

When LICQ fails, we have to assume Slater's condition [93] holds. The Slater's condition imposes restrictions on the feasible set and requires the existence of a point in the feasible set such that inequality constraints are strictly smaller than 0. The Slater's condition is not strict and only fails to hold for pathological cases in our problem. One pathological example is due to velocity limit and time limit, the drone has to be at the maximum velocity to travel from one end to another. Considering a 1D single corridor case starting from 0 to 1 within 1 seconds. The velocity limit of 1 m/s requires all velocity inequalities constraints to be active and the feasible set has no interior. This case, however, is close to infeasibility so it rarely occurs. Besides, this case is unstable and small perturbation to y solves the issue. For convex problems, Mangasarian-Fromovitz Constraint Qualification (MFCQ) [56] holds if Slater's condition holds. Then the optimal cost of the lower problem is directional differentiable [56, Theorem 7.3] in any direction and the directional derivative is given by

$$\nabla_y J^*(\bar{y}, z) = \min_{\bar{c} \in S(\bar{y})} \max_{\mu \in M(\bar{c}, \bar{y})} \nabla_y L(\bar{c}, \bar{y}, \mu) z \tag{2.18}$$

where z with $\|z\| = 1$ is any direction; $S(\bar{y})$ is the set of minimizers which is singleton in our case due to uniqueness of the optimal solution; μ is the Lagrangian multipliers vector and $M(\bar{c}, \bar{y})$ is the set of valid multipliers that satisfy KKT conditions. In our problem, the optimum is unique due to SSOSC so we can remove the minimum operator. The directional derivative is obtained by essentially solving a linear program to compute the multipliers. According to the proof within [56, Theorem 7.3], M is compact if MFCQ holds. Since

constraints are linear, M is essentially a polyhedron defined by equality and inequality constraints of KKT equations, so is the set of values of $\nabla_y L(\bar{c}, \bar{y}, \mu)$ due to its linear dependency on μ. For all the vertices, there exists a z such that the maximum is obtained at the vertex due to convexity. In that direction, $\nabla_y L(\bar{c}, \bar{y}, \mu)$ computes the exact gradient. So according to the definition in Eq. (2.15), $\nabla_y L(\bar{c}, \bar{y}, \mu)$ is within the subdifferential. So the polyhedron of $\{\nabla_y L(\bar{c}, \bar{y}, \mu) | \mu \in M(\bar{c}, \bar{y})\}$ is a subset of the Clarke differential, and so is the gradient computed by any valid μ.

In fact, $\nabla_y L(\bar{c}, \bar{y}, \mu)$ may be unique for all $\mu \in M(\bar{c}, \bar{y})$ such as when the non-dependent constraints do not depend on y. In that case, the exact gradient is computed. One example in our problem is when two corridors share a face (denoted as the plane $x = b$) and the constraints include $x_i \leq b, x_j \geq b, x_i = x_j$ where x_i and x_j is the last and first control points of the two corridors, respectively. The first two constraints are the control points that are within their corresponding corridors and the third one is path continuity. At optimum these 3 constraints are always linearly dependent but do not affect the gradient computation since they do not depend on the time allocation. Moreover, one redundant inequality constraint is removed during the pre-solve process of the optimizers.

2.4.4 Constraints in the Outer Optimization

This section shows that even with constraints in the outer optimization, the theory in Sec. 2.4.1 can be applied. The outer optimization in the Soft-Time variant as in Eq. (2.8) has the form

$$\begin{aligned} \underset{y}{\text{minimize}} \quad & J(y) = c^\star + w \mathbb{1}^T y \\ & y \geq 0. \end{aligned} \tag{2.19}$$

The problem is essentially unconstrained at optimum. The reason is that as $y_i \to 0$, the cost function approaches ∞ and non-positive durations can be rejected during line search. So the constraint $y \geq 0$ can be ignored in analysis.

The Hard-Time variant as in Eq. (2.7) has the form

$$\begin{aligned} \underset{y}{\text{minimize}} \quad & J(y) = c^\star(y) \\ & y \geq 0 \\ & \mathbb{1}^T y = T_c \end{aligned} \tag{2.20}$$

with fixed total traversal time T_c.

Here we use a projected (sub)gradient method that computes the (sub)gradient $g \in \mathbb{R}^n$ and projects it onto the hyperplane $\mathbb{1}^T g = 0$ so an update does not change the total traversal

time. The projection operator is given by

$$\mathcal{P}g = g - (\mathbb{1}^T g/n)\mathbb{1} = (I - \frac{1}{n}\mathbb{1}\mathbb{1}^T)g$$

and the update rule is

$$y' \leftarrow y - \alpha\mathcal{P}g = y - \alpha(I - \frac{1}{n}\mathbb{1}\mathbb{1}^T)g. \tag{2.21}$$

We show that this is equivalent to (sub)gradient descent on an unconstrained problem:

$$\begin{array}{ll} \underset{s}{\text{minimize}} & \tilde{J}(s) = c^\star(y_0 + As) \\ & y_0 + As \geq 0 \end{array} \tag{2.22}$$

where $s \in \mathbb{R}^{n-1}$; y_0 is the initial time allocation which satisfies $\mathbb{1}^T y_0 = T_c$; and $A \in \mathbb{R}^{n\times(n-1)}$ is a matrix forming the orthonormal basis of the null space of $\mathbb{1}$. A can be obtained by the singular value decomposition $\mathbb{1} = USV^T$, with $U = [1]$, $S = [\sqrt{n}, 0, \ldots, 0]$, and $V \in \mathbb{R}^{n\times n}$ is an orthogonal matrix with first column $\mathbb{1}/\sqrt{n}$ and the remaining rows equal to$A$, that is $V = \left[\begin{array}{c|c} \mathbb{1}/\sqrt{n} & A \end{array}\right]$.

The value y corresponding to an iterate s is given by $y_0 + As$. Using the same argument as in the Soft-Time variant, we can show the inequality constraint does not affect convergence. Each iterate of gradient descent satisfies the constraint $\mathbb{1}^T(y_0 + As) = T_c$. Using chain rule, the (sub)gradient of $\tilde{J}$ w.r.t. s is $A^T g$ where g is a (sub)gradient of J at $y_0 + As$. The update rule is thus

$$s' \leftarrow s - \alpha A^T g$$

so the equivalent time allocation update is

$$y' \leftarrow y_0 + As' = y_0 + As - \alpha AA^T g = y - \alpha AA^T g. \tag{2.23}$$

Using the orthogonality of V we have

$$I = VV^T = \begin{bmatrix} \mathbb{1}/\sqrt{n} & A \end{bmatrix} \begin{bmatrix} \mathbb{1}^T/\sqrt{n} \\ A^T \end{bmatrix} = (\mathbb{1}\mathbb{1}^T)/n + AA^T$$

which demonstrates that $AA^T = I - \frac{1}{n}\mathbb{1}\mathbb{1}^T$. Hence, the unconstrained update rule (2.23) is equivalent to (2.21).

2.5 EXPERIMENTS

Our algorithm, which is released as an open-source package[3], is implemented in Python. The QP solvers are called with interfaces to C++ libraries. Since solving QP dominates the running time, our reported computation time is similar to a pure C++ implementation with some overhead from Python.

2.5.1 Numerical Experiments

We evaluate our method on random instances of "indoor flight". We take the indoor building environment from [96], extrude it along z axis, and discretize the z direction into 5 cells. We then select some rows and columns in the image as shown in Fig. 2.3 and fill the bottom 3 or top 3 cells (along z axis) as obstacles according to the color. The start and goal are randomly sampled in the free space so in most cases there is movement along z axis. The environment is shown in Fig. 2.3. The complexity of the environment allows us to generate problems with more than 40 segments. 200 problems are randomly generated. One example is shown in Fig. 2.4. We note that despite we are exclusively using axis-aligned boxes as safe corridors for simplicity, convex polyhedrons may yield less conservative results and our method is able to handle it as well. But there is a trade-off between the complexity of corridors and optimality of the solution.

Figure 2.3: The 2d floor plan used to generate random test problems. Here the orange and light green lines shows where obstacles along z axis are placed.

We solve the Hard Time variant as in Eq. (2.7), with the initial guess of time allocation and fixed total traversal time computed from the heuristic introduced by Gao et al [40].

[3] https://github.com/OxDuke/Bilevel-Planner

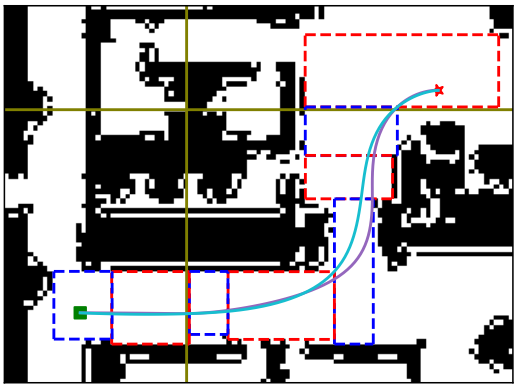


Figure 2.4: A random problem instance. The start (square), goal (star), and generated corridor (dashed boxes) are shown. The purple and cyan curve are the optimal trajectories with the initial and final time allocations.

In this section, the QP solver is Gurobi [4], a commercial interior-point QP solver. We set a major iteration limit of 50 to limit the total computation time. The velocity limit is set as 2 m/s. To handle cases with infeasible initial time allocation, we simply multiply the time allocation by a scalar until feasibility is obtained. As a benchmark, we compare our analytic gradient with a finite-difference approximation method under the same bilevel optimization framework. Another method in comparison is to optimize the control points and time allocation simultaneously using nonlinear optimizer SNOPT [97]. Finally, we compare with direct collocation method [80] which discretize the trajectory, formulate an NLP and solve it using SNOPT. These experiments are carried out on a workstation with a 3.30 GHz Intel Xeon W-2155 processor, using only one thread.

Finite Difference vs Analytic Gradient In this section, we solve the same problem set as before and compare the analytic gradient from Lagrangian multipliers defined in this paper with finite difference approximation, where one gradient computation requires additional n QPs being solved. The average performance is shown in Tab. 2.1. Clearly, analytical gradients are not only much faster, but also help the optimizer converge to a better solution because of their higher precision.

Comparison with Joint Optimization Joint optimization directly solves Eq. (2.7) as an NLP using SNOPT [97], a general nonlinear solver for sparse, large-scale problems. On the contrary, bilevel optimization decouples temporal and spatial variables and solves them hierarchically. We provide analytic gradients to SNOPT for solver robustness and explore

[4] `https://www.gurobi.com/`

Table 2.1: The Mean Total Computation Time, Average Major Iteration Time and Normalized Cost (final cost over the cost from heuristic assignment) of Finite Difference (FD) w.r.t. Analytic Gradient (AG)

	Total Time (s)	Avg. Iter. Time (s)	Normalized Cost
FD	15.403	0.497	0.109
AG	0.874	0.024	0.068

problem sparsity to the best of our ability. We initialize SNOPT with the unrefined time allocation and the spline coefficients computed in the first QP solve. All the stopping criteria are set to default except the optimality tolerance is set to 1×10^{-3}.

We obseved that joint optimization is susceptible to the strong non-linearity of the problem and only 11 out of 200 problems converged to a feasible solution, as shown in Tab. 2.2. Due to the low success rate, it's clearly not suitable for the application. SNOPT tends to terminate prematurely without converging, and often moves to an infeasible point even though it starts from a feasible solution. We believe this is because the joint spatial and temporal NLP is ill-conditioned. The QP objective function exhibits high-order dependence on timing, and some spatial constraints are very sensitive to the high-order spline coefficients. On the other hand, in the bilevel formulation, the ill-conditioned problem is handled by convex solvers, which are known to be more robust. We also note that SNOPT has no guarantee on obtaining a feasible solution while our approach can be terminated at any time and return a feasible solution. Usually nonlinear optimizers require an initial guess close to the optimum values to converge. With the same initial guess of time allocation, our method is able to make progress towards optimum while SNOPT moves from a feasible initial guess to non-feasible solutions.

Comparison with Direct Collocation We also compared our proposed method with the Direct Collocation method (DC) [80] to solve problem (2.7) as an alternate NLP formulation. DC optimizes over discretized states $[p(t), \dot{p}(t), \ddot{p}(t)]$ where $p(t)$ is position and control is $u \equiv \dddot{p}(t)$ at each collocation grid point $t_0, \ldots, t_N$. To adjust the timing of each segment, the initial and final times of each segment are introduced as additional decision variables. The state and control trajectories are optimized simultaneously with segment times. Each segment has a fixed grid size. See Appendix 2.7.2 for the details.

Once again, we use SNOPT to solve the NLP. The major iteration limit and total iteration limit are set as 500 and 5000, respectively, to keep the total computation time manageable. Due to the different NLP formulation, the final objective value from DC cannot be directly compared to other approaches so we just compare success rate instead. Direct collocation

does not perform well on this problem and only 44 out of 200 problems converge to an optimal solution, as shown in Tab. 2.2. We observed that with a higher iteration limit, DC can achieve a higher success rate. However, the average computation time with the current settings is already 7.12 s, which is unsuitable for responsive quadrotor flight.

Table 2.2: Success Rate of Bilevel Optimization, Joint Optimization, and Direct Collocation

Bilevel Optimization	Joint Optimization	Direct Collocation
200/200	11/200	44/200

2.5.2 Scalability Study

For complex environments, the number of safe corridors may be as large as around 50 and thus imposes a challenge to the computational efficiency. We perform an empirical scalability study to see how our framework performs with increasing number of segments using the same problems in the last section. We also study the effect of QP solvers and compare two types of QP solvers: active-set solver Sqopt [98] and interior-point solver Gurobi. Both solvers are designed for sparse and large-scale problems. Our previous test suite [99] showed that Sqopt performs better when the average number of segments is below 10. In this paper we want to study how it performs when the problem has more segments. We use the same setting as the last section with the two QP solvers.

It turns out the performance of our algorithm in terms of cost function is quite consistent with the two solvers. Numerical errors result in slight difference in the progress of the algorithm in each solver. We note that we set an iteration limit of 50 and around 60 problems out of 200 are terminated after reaching the iteration limit. On average each iteration requires 1.2 QPs being solved. This also indicates the total computation time is roughly proportional to the average time of each QP solving. However, the computation time is quite different for the two solvers, indicating different scalability in QP solving time.

The computation time is different since Sqopt and Gurobi have different scalability with the number of segments which is proportional to the number of optimization variables and constraints. Gurobi implements an interior-point method so its iteration number is roughly constant. In each iteration a sparse block-diagonal matrix is factorized which takes time linearly to the number of segments. Sqopt implements the active-set method which does not scale well since it takes more iterations to identify the correct active set. However, it is faster than Gurobi when the number of segments is below 10, which is consistent with the results in [99]. If finite-difference is used to estimate the gradient, the algorithm has quadratic and cubic scalability when the QP solver is Gurobi and Sqopt, respectively.

The total computation time scales similarly to average QP times since similar numbers of QPs are solved, as shown in Fig. 2.5. The total computation time scales linearly when Gurobi is the QP solver. The total computational time shows the potential of applying our algorithm in real-time since the major improvement of cost functions occurs in the first few iterations, evidenced in Fig. 2.6. For large problems with limited computation time, a smaller iteration limit has to be used, although it does not affect the cost function too much.

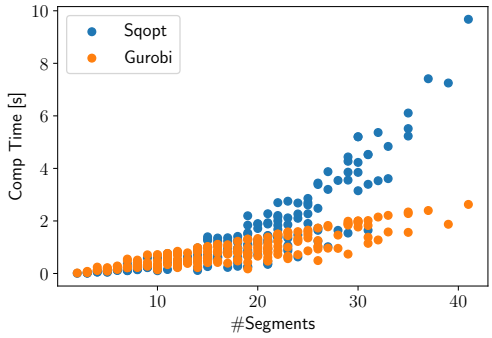


Figure 2.5: The total computation time of our algorithm as a function of the number of segments in random problems.

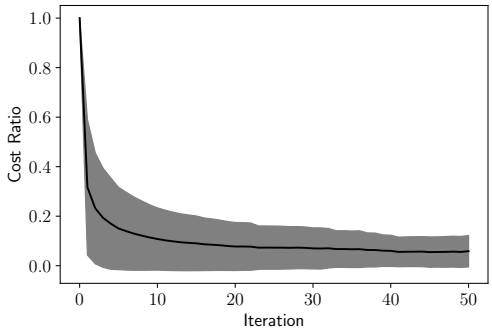


Figure 2.6: The profile of the average (solid line) and standard deviation (shaded) of the cost ratio over the initial cost. The cost ratio is computed by dividing the cost at current iteration by the initial cost.

2.5.3 Effect of Subgradient Step

Although the numerical experiments above are conducted on challenging 3D problems, we found that the subgradient step of Alg. 1 is only triggered once. To better examine the effect of subgradient steps, we designed a problem set that triggers subgradient steps more frequently. These variants project the environments into 2D to eliminate vertical movement,

and velocity limits are disabled. 808 random problem instances of this form were generated, and we compared Alg. 1 with and without Lines 9–12.

The results are shown in Fig. 2.7, with both Sqopt and Gurobi as the underlying QP solver. A subgradient step was taken in 15 problems using Squopt, and in 65 problems using Gurobi. The discrepancy between solvers is due to the slightly different gradient estimation, which may lead to different algorithmic behavior overall. The plot combines results from subgradient steps taken with either QP solver. For many problems, the subgradient step decreases the cost substantially. However, it does pay a price in terms of increased computation time. The reason is that the gradient-only variant stalls out more quickly, while the subgradient variant continues to optimize and make progress.

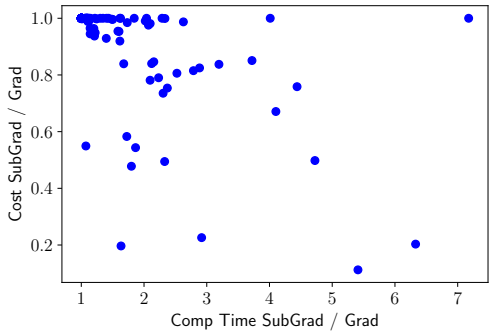


Figure 2.7: Enabling subgradient descent can improve cost when pure gradient descent gets stuck. The x and y axis are the ratios of computation time and cost function, respectively, of Alg. 1 with subgradient descent enabled vs the disabled variant. Lower is better for both axes.

2.5.4 Physical Experiments

We validate our planner on an indoor obstacle avoidance scenario using a commercially available small-scale quadrotor, Crazyflie 2.1[5]. A video that compiles all the physical experiments is provided as supplementary material.

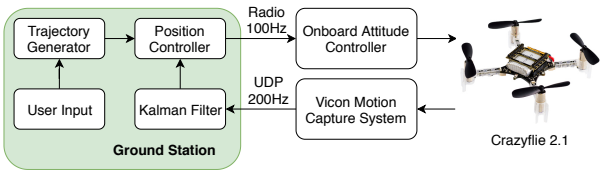


Figure 2.8: Physical quadrotor system setup

[5]https://www.bitcraze.io/

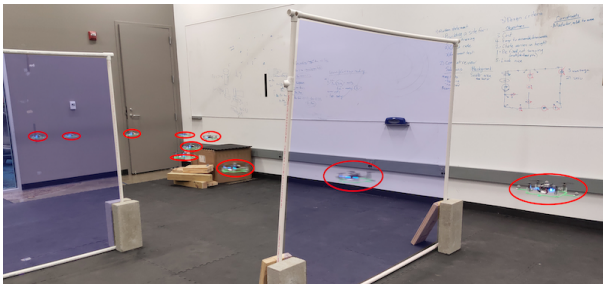


Figure 2.9: Obstacle layout for physical quadrotor experiments, with the space within the frames (tinted blue) are treated as obstacles. Evenly spaced frames from the executed trajectory are overlaid, with the quadrotor circled in red.

The system setup is shown in Fig. 2.8. The position of the quadrotor is captured by the Vicon[6] motion capture system and transmitted to ground control station using Ethernet at 200Hz. The raw data stream from Vicon goes through a Kalman filter and then serves as feedback for a position controller on the ground control station. The position controller is a feedfoward-feedback controller, with the feedforward term computed from the reference trajectory and its time derivative thanks to the differential flatness property, and the feedback term computed from a proportional-integral-differential (PID) controller.

As shown in Fig. 2.9, we set up two frames as walls, and considered them as obstacles in our algorithm. We run 3 experiments to demonstrate the effectiveness and real-time capability of our algorithm. Admittedly, in these physical experiments the number of segments is not large. For problems with more segments, a smaller iteration limit can be used without losing too much optimality. Indeed, the property of any-time feasibility of our algorithm is suitable for challenging problems with many segments since the user can set arbitrary number of iterations.

Comparison to time allocation heuristics Our algorithm plans a faster trajectory while achieving the same control effort (jerk) as Gao el al. [40], which uses time allocation heuristics. The quadrotor starts at some initial position and is asked to travel to a target position chosen by a human operator using an Rviz[7] interface while avoiding obstacles. Both methods are set up to solve the Hard Time variant (2.7), but after running our algorithm, we scale the total traversal time T until the jerks of two trajectories becomes the same. Table. 2.3 indicates that our algorithm yields a 12% shorter and 18% faster trajectory with equivalent jerk.

[6]https://www.vicon.com/
[7]http://wiki.ros.org/rviz

Table 2.3: Comparing against heuristic time assignment

Method	Length	Traversal Time	Jerk
Ours	**5.15** m	**4.36** s	39
Gao et al.	5.82 m	5.32 s	39

Controlling aggressiveness using time penalty w Next we show the Soft Time variant (2.8) can handle objectives with various time penalties to control aggressiveness. Results are summarized in Table 2.4. These indicate that, as expected, when the weight penalty increases, trajectories become faster and more jerky. Also, computation time is not significantly affected by the weight parameter.

Table 2.4: Comparing motion aggressiveness parameters

Weight (w)	Traversal Time	Computation time	Jerk
10	5.60 s	10.8 ms	11.2
20	4.96 s	10.7 ms	19.9
40	4.42 s	10.7 ms	36.1
80	4.01 s	9.3 ms	64.7

Tracking a dynamic goal The final experiments show the algorithm running in real-time, where the quadrotor tracks a dynamic goal moved by a human while avoiding obstacles. The goal is tracked by Vicon and is used for replanning at 3Hz. Trajectories are generated using the Soft Time variant (2.8) with weight $w = 80$. In these experiments, each optimization takes less than 15 ms.

2.6 CONCLUSION

We presented a novel bilevel optimization approach to UAV trajectory optimization, which analytically calculates the gradient of the objective function w.r.t. temporal variables. The optimization method takes into account the non-smoothness of the upper-level optimization problem. Our results show that this approach achieves real-time performance and higher quality trajectories than state-of-the-art heuristics. Our method can handle both hard time variant with fixed total time and soft time variant where the weight is used to adjust trajectory aggressiveness. It can be useful in multiple contexts such as formation flight and tracking dynamic targets.

Future work may include accelerating the gradient descent by exploiting the structure of

the problem. For example, acceleration may be achieved through using Newton or Quasi-Newton methods. We are also interested in studying extensions of the bilevel optimization approach to other robotic applications like autonomous vehicles and legged locomotion.

2.7 APPENDICES

2.7.1 Proof of SSOSC

We prove the second-order sufficient conditions (SSOSC) holds in TOBC where jerk is minimized amongst Bézier curves of order 6. For brevity we drop dependency on y.

SSOSC [65] states that the Hessian of the Lagrangian evaluated at the optimal point is positive definite on the null space of the gradients of all the active constraints, i.e.,

$$w^T \nabla^2_{cc} L(c^\star, \lambda^\star, \nu^\star) w > 0,\ \forall w \neq 0 \,\text{s.t.}\, G_{\mathcal{A}} w = 0; Hw = 0$$

where $G_{\mathcal{A}}$ collects the rows of active inequality constraints and $L(\cdot)$ is the Lagrangian:

$$L(c, \lambda, \nu) = c^T P c + \lambda^T G c + \nu^T H c,$$

where λ and ν are the associated Lagrange multipliers.

Our proof shows that there does not exist any $w \neq 0$ that lies in the null space of P and H simultaneously, which proves SSOSC because $\nabla^2_{cc} L(c^\star, \lambda^\star, \nu^\star) = P$, and since P is a symmetric positive semi-definite matrix, $w^T P w = 0$ implies $Pw = 0$. (Note that we ignore $G_{\mathcal{A}}$, so the proof holds regardless of which inequality constraints are active.)

Since each of the x, y, z dimensions can be decoupled, we show the proof in one dimension without loss of generality. Matrix P is block diagonal, with blocks denoted $P_i \in \mathbb{R}^{7\times 7}$, whose entries depend on the duration of the corresponding segment. It can be be shown that

$$\mathcal{N}_{P_i} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \\ 1 & 5 & 25 \\ 1 & 6 & 36 \\ 1 & 7 & 49 \end{bmatrix}$$

is a basis for the null space of P_i, regardless of segment duration. The physical meaning

for each column of $\mathcal{N}_{P_i}$ is not moving at all, moving with constant velocity, and moving with constant acceleration, respectively, since these motions induce zero jerk. Then, for a problem with n segments, the matrix

$$\mathcal{N}_P = \begin{bmatrix} \mathcal{N}_{P_i} & & \\ & \ddots & \\ & & \mathcal{N}_{P_i} \end{bmatrix}$$

is a basis of the null space of P, and has rank $3n$.

The linear equality constraints include initial and final position, velocity, and acceleration which accounts for 6 constraints. Continuity of position, velocity, and acceleration at conjunction points provide another $3(n-1)$ constraints for a problem with n segments. The linear equality constraint matrix H is defined as

$$H = \begin{bmatrix} L_1 & 0 & & & & & & & \\ & 0 & R_1 & -L_2 & 0 & & & & \\ & & & & 0 & R_2 & & & \\ & & & & & & \ddots & & \\ & & & & & & & -L_n & 0 \\ & & & & & & & 0 & R_n \end{bmatrix}. \tag{2.7.1}$$

with 3×3 blocks:

$$L_i = \begin{bmatrix} 1 & & \\ -d/\Delta t_i & d/\Delta t_i & \\ d(d-1)/\Delta t_i^2 & -2d(d-1)/\Delta t_i^2 & d(d-1)/\Delta t_i^2 \end{bmatrix},$$

$$R_i = \begin{bmatrix} 0 & & 1 \\ 0 & -d/\Delta t_i & d/\Delta t_i \\ d(d-1)/\Delta t_i^2 & -2d(d-1)/\Delta t_i^2 & d(d-1)/\Delta t_i^2 \end{bmatrix},$$

and each 0 block is of size $3 \times (d-5)$. Here, Δt_i is the time allocated to the i'th segment and $d = 6$ is the spline degree.

Now we show that $H\mathcal{N}_P$ has full column rank. First, split $\mathcal{N}_{P_i}$ into blocks $[Q_1^T, Q_2^T, Q_3^T]^T$ where $Q_1, Q_3 \in \mathbb{R}^{3\times 3}$ and $Q_2 \in \mathbb{R}^{1\times 3}$. Note that Q_1 and Q_3 are square matrices of full rank.

With this notation we express the product

$$H\mathcal{N}_P = \begin{bmatrix} L_1Q_1 & & & \\ R_1Q_3 & -L_2Q_1 & & \\ & R_2Q_3 & \ddots & \\ & & \ddots & -L_nQ_1 \\ & & & R_nQ_3 \end{bmatrix}.$$

Observe that each block is non-singular because it is a product of two square matrices of full rank. The upper $3n$ by $3n$ part is full column rank since each block is non-singular. Therefore, $H\mathcal{N}_P$ has full column rank.

Consequently, any $w \neq 0$ in the nullspace of P can be expressed as $w = \mathcal{N}_P v$ with $v \neq 0$, which shows that $Hw \neq 0$. Hence there does not exist $w \neq 0$ that lies in the null space of P and H simultaneously, which proves SSOSC as desired.

2.7.2 Direct Collocation Implementation

This section provides details of the direct collocation (DC) method implemented for our numerical experiments. Our implementation follows the standard *Hermite-Simpson* collocation method, see e.g. Betts [80, Section 5.1]. DC works by discretizing a continuous-time trajectory optimization problem into a NLP with discretized states and controls as decision variables. However, direct collocation is rarely used in UAV trajectory optimization since it does not take advantage of the differential flatness and is usually used directly with the system's actual dynamics equations. In order to use DC to minimize trajectory jerk, we define the system state $x(t) = [p(t), \dot{p}(t), \ddot{p}(t)]^T \in \mathbb{R}^9$ as a stacked vector of position $p(t) \in \mathbb{R}^3$, velocity $\dot{p}(t) \in \mathbb{R}^3$ and acceleration $\ddot{p}(t) \in \mathbb{R}^3$. The control input $u(t)$ is chosen to be the jerk $u(t) = \dddot{p}(t) \in \mathbb{R}^3$.

The system dynamics can be written as:

$$\dot{x}(t) = f\big(x(t), u(t)\big) = Ax(t) + Bu(t) \tag{2.7.2}$$

where

$$A = \begin{bmatrix} 0 & I_{3\times3} & 0 \\ 0 & 0 & I_{3\times3} \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ I_{3\times3} \end{bmatrix}.$$

Consider a safe corridor with N segments, we set up a N-phase trajectory optimization

problem. For the i'th phase, the trajectory is constrained to stay in the i'th convex region. Continuity constraints up to acceleration are applied between consecutive segments. This formulation matches with TOBC. The decision variables include:

1. Duration of each phase: $\Delta t_i, i = 1, \ldots, N$,
2. Discretized state trajectory of the UAV for each phase: $x_i(t), t \in [0, \Delta t_i], i = 1, \ldots, N$,
3. Discretized control trajectory for each phase: $u_i(t), t \in [0, \Delta t_i], i = 1, \ldots, N$,

and the cost function is

$$J = \sum_{i=1}^{N} \int_0^{\Delta t_i} \|u_i(\tau)\|^2 d\tau + w\Delta t_i. \tag{2.7.3}$$

The rest follows the standard DC formulation [80]

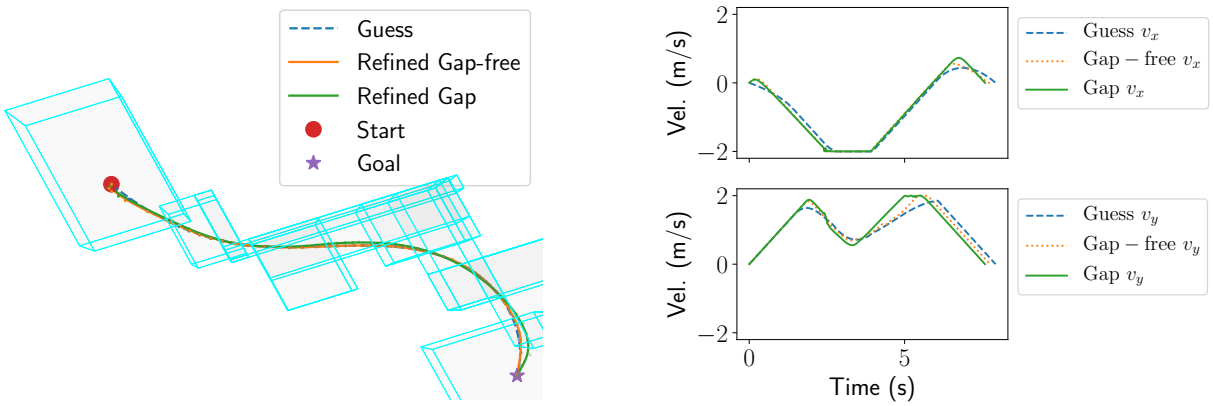
CHAPTER 3: ENHANCING BILEVEL OPTIMIZATION FOR UAV TIME-OPTIMAL TRAJECTORY USING A DUALITY GAP APPROACH

In Chapter 2 the decomposition of spatial and temporal variables is used in a bilevel optimization framework. It takes advantage of the structure that with temporal variables fixed, the spatial variables are solved with convex optimization. In this chapter, the bilevel optimization framework is used to solve the minimum-time trajectory for UAVs. With spatial variables fixed, the temporal variables are solved by convex optimization for minimum-time trajectory. Similar to the improvements in Chapter 2, it enables optimization of the spatial trajectory while the baseline method keeps it fixed. Moreover, in this problem it is simply too computationally expensive to use finite-difference to compute gradients given the high-dimensionality of the parameterization of spatial trajectory. The analytic gradient is again computed using Lagrangian multipliers from lower optimization and is key to the efficiency of the framework. However, a different approach is proposed to handle non-smoothness. The duality gap approach is introduced to the lower problem which is mathematically equivalent to adding log barriers to the inequality constraints. With the log barrier, inequality constraints are never active (an inequality constraint is active when equality holds) and switches of constraint activeness are eliminated, so is non-smoothness. Warm-start of lower optimization is enabled to further improve computational efficiency. This framework has shown improvements in reliability, efficiency and optimality compared with ordinary trajectory optimization approaches and bilevel optimization without duality gap. [1]

[1] This chapter is reproduced from Gao Tang, Weidong Sun, and Kris Hauser, "Enhancing bilevel optimization for uav time-optimal trajectory using a duality gap approach". In 2020 IEEE International Conference on Robotics and Automation (ICRA).

3.1 INTRODUCTION

Time-optimal trajectory optimization is important for drones, vehicles, and industrial manipulators to complete tasks efficiently, but it is inherently difficult even for linear systems due to its bang-bang control structure. Typical approaches to trajectory optimization, such as Direct Collocation (DC), can handle a wide variety of costs, state constraints, and control constraints by converting the problem into a nonlinear programming (NLP) optimization, which can be solved by numerical techniques [80]. For time minimization, the state and control are functions of time and solved simultaneously with the optimal trajectory time. This approach introduces significant nonlinearity in the dynamics and non-convexity in constraints, so there is no guarantee that an optimal, or even feasible solution is obtained. Alternatively, a two-stage approach approximately solves the problem by optimizing the geometric path separately from the velocity profile, which exploits the speed and robustness of Time Optimal Path Parameterization (TOPP) [53, 100]. The drawback of two-stage optimization is that the path is fixed after the first stage, while it is possible to further refine it to reduce trajectory time. Following the approach in Chapter 2, the bilevel optimization framework is applicable here as well. The gradients of the optimal traversal time with respect to the path can be computed analytically, allowing the outer optimization to be addressed as a standard NLP while the convexity of the TOPP problem lends itself to efficient and reliable Interior-Point Methods (IPMs).



(a) Spatial shape

(b) Velocity profile

Figure 3.1: (a) Initial and refined trajectory for a randomly generated problem. The feasible space of the environment is split into convex boxes in which the path must lie, and an initial minimum-jerk trajectory is refined to decrease trajectory traversal time with velocity and acceleration limits. With the proposed gap technique, the trajectory converges more quickly toward an optimum. (b) It also achieves a lower execution time compared with a gap-free approach.

This chapter makes further improvements to the bilevel optimization framework. In [101], the efficiency is limited by the slow convergence rate of the outer optimization, and warm-starting of the inner IPM is not effective, because the prior optimum reaches the boundary of inequality constraints and is far from the central path for the new problem. Furthermore, the cost function in outer optimization is continuous but non-smooth due to changes of active inequalities in the inner optimization. Function smoothness is important for the convergence rate of gradient based methods such as the difference between gradient and sub-gradient methods [93]. To handle these problems, we introduce a *duality gap* which keeps the solution away from the boundary, and deliberately do not solve the optimization to optimality. With a controlled duality gap, inequality constraints are never active and thus there is no switch of active constraint and the solution of IPM is never at the boundary. This trick helps to achieve 1) warm-start of inner optimization and 2) a smoother outer cost function landscape, which significantly accelerates inner and outer optimization, respectively. We prove that solving to a desired duality gap is equivalent to the log-barrier method for penalizing inequality constraints, and the gradient computation time is equivalent to the method of [101].

Numerical experiments on an aerial vehicle with velocity and acceleration limits demonstrate the robustness, efficiency, and optimality of our approach over standard bilevel optimization, direct collocation, and simultaneous optimization of the path and velocity profile, using state-of-the-art NLP solvers IPOPT [102] and SNOPT [103].

3.2 RELATED WORK

A general overview of motion planning for autonomous vehicles can be found in [3]. The time-optimal problem for vehicles has been extensively studied [60, 104, 105, 106, 107, 108, 109]. The brittleness of spatio-temporal trajectory optimization using direct collocation has been noted by other researchers, leading to a search for alternative, more robust techniques.

3.2.1 TOPP Solvers

Time-Optimal Path Parameterization (TOPP) seeks admissible control inputs that minimizes the traversal time of a pre-specified path subject to constraints such as system dynamics, control actuation, and velocity limit. Efficient approaches to it include: (1) convex optimization (CO) [53, 59, 108], (2) numerical integration (NI) [100, 110], and (3) reachability analysis (RA) [111]. Our TOPP solver solves the nonlinear optimization problem directly and exploiting sparsity in the KKT solver [101] for efficiency. Since CO methods

are used, Lagrange multipliers are by-product and allow for calculation of the gradient of the minimum time with respect to the path. No other methods besides CO can provide such gradient information. Similar to [101], we use primal-dual IPM to solve the nonlinear CO directly, instead of converting to SOCP [53] or using Sequential Linear Programming [59] or primal barrier approach [108]. The sparsity of KKT solver is exploited to achieve linear scalability as done in [101].

3.2.2 Two-stage Optimization

TOPP with a half-car model is studied by Velenis and Tsiotras [109], and is used as a submodule in Kapania *et al.* [60]. There is work trying to remove the limitation of the two-stage approach by updating the geometric path too. Kapania et al. presents a two-stage iterative method for generating time-optimal trajectories through a race course. However, they are minimizing path curvature to update the path which is different from minimizing lap time. By contrast, our approach updates the path with a goal of minimizing traversal time. Similarly Gao et al. [41] uses a two-stage approach to compute aggressive trajectories for UAVs. The temporal variables are optimized to minimize traversal time similar to TOPP but the spatial trajectory is optimized to minimize the total jerk. These two sets of variables are iteratively optimized alternatively and it is unclear what the solution is converging to.

3.3 PROBLEM FORMULATION

3.3.1 Time-Optimal Motion Planning Problem

A system has generalized configuration $q \in \mathbb{R}^n$ and operates in an environment represented by $\mathcal{X}_{\text{free}} \subseteq \mathbb{R}^n$, which denotes all the collision-free configurations. The problem is to find a trajectory in $\mathcal{X}_{\text{free}}$ from a start configuration q_s to a configuration in a goal set $\mathcal{X}_{\text{goal}} \subseteq \mathcal{X}_{\text{free}}$ in a minimum amount of time while respecting constraints such as dynamics, collision avoidance, and state and control bounds. The problem can be mathematically formulated as:

$$\begin{aligned} \underset{q(t),u(t)}{\text{minimize}} \quad & T \\ \text{subject to} \quad & q(t) \in \mathcal{X}_{\text{free}}, \quad \forall t \in [0, T] \\ & q(0) = q_s, \\ & q(T) \in \mathcal{X}_{\text{goal}}, \\ & d(q(t), \dot{q}^2(t), \ddot{q}(t), u(t)) \leq 0, \quad \forall t \in [0, T] \end{aligned} \tag{3.3.1}$$

where $u(t)$ is the control input, T is the trajectory duration, and $d(\cdot)$ collects various constraints mentioned before.

This problem is difficult to solve directly due to its non-convexity since it has to optimize path $x(t)$ and time T simultaneously. Our bilevel approach optimizes path and time parameterization hierarchically and use the fact that if $x(t)$ is fixed, a proper parametrization allows convex optimization of temporal variables.

3.3.2 Path Representation

A feasible *geometric path* is a continuous curve $p : [0, 1] \to \mathcal{X}_{\text{free}}$ that connects the start configuration q_s and goal configuration q_g such that $p(0) = q_s, p(1) = q_g$. We represent paths using piecewise Bézier curves with C^2 continuity for each DOF to simplify collision avoidance similar to Chapter 2. A decomposition of $\mathcal{X}_{\text{free}}$ is done as well.

For a space decomposed into K convex regions, the path p of order m is represented by a vector of length $n(m + 1)K$, $p = [c_0^1|c_1^1|\cdots|c_m^1|c_0^2|c_1^2|\cdots|c_m^K]^T$ where c_i^k is the ith control point for the kth segment.

If h_i is the number of hyperplanes bounding the i-th convex polyhedron, a total of $n(m + 1)\sum_{i=1}^{K} h_i$ linear inequality constraints are sufficient to guarantee collision avoidance. This is represented by an inequality like (with properly chosen G_p and g_p):

$$G_p p \leq g_p. \tag{3.3.2}$$

By the property of Bézier curve, it is sufficient to guarantee the curve is inside the polyhedron by only constraining the control points [73].

We impose C^2 continuity constraints on neighboring segments to enforce path smoothness. This is achieved by constraining the control points. Also, the path should obey terminal constraints, i.e. $c_0^1 = q_s, c_m^K = q_g$. These linear equality constraints are in the format of:

$$H_p p = h_p. \tag{3.3.3}$$

with properly chosen H_p and h_p.

3.3.3 Time-Optimal Path Parameterization

TOPP aims to find a time parameterization to a geometric path, so that the traversal time is minimized while satisfying all constraints. Pioneered by [53, 59], it is formulated as a convex optimization problem under appropriate assumptions. Here we give a brief review.

A *time parameterization* is a monotonously increasing scalar function $s(t) : [0, T] \to [0, 1]$, where T is the traversal time of the path. The *trajectory* is the time-parametrized geometric path and represented as $q(t) = p(s(t)) : [0, T] \to \mathcal{X}_{\text{free}}$. By chain rule, we have

$$\dot{q}(t) = p'(s)\dot{s}(t), \quad \ddot{q}(t) = p'(s)\ddot{s}(t) + p''(s)\dot{s}^2(t) \tag{3.3.4}$$

where $\dot{\square}$ and $\square'$ denotes derivative w.r.t. time and s, respectively.

In TOPP, values of $p'(s)$ and $p''(s)$ are known and not altered when the geometric path is given, the time allocation $s(t)$ has to be optimized. We introduce two variables $a(s) = \ddot{s}, \quad b(s) = \dot{s}^2$ which satisfy the following relationship

$$\dot{b}(s) = b'(s)\dot{s} = \frac{d(b(s))}{dt} = 2\dot{s}\ddot{s} = 2a(s)\dot{s}$$

or more simply,

$$b'(s) = 2a(s). \tag{3.3.5}$$

The traversal time T can be written in terms of $b(s)$ as

$$T = \int_0^T 1dt = \int_{s(0)}^{s(T)} \frac{1}{\dot{s}} ds = \int_0^1 \frac{1}{\dot{s}} ds = \int_0^1 \frac{1}{\sqrt{b(s)}} ds. \tag{3.3.6}$$

To get a convex problem, we follow [59, 112] and discretize $s(t)$ into N segments: $\{b_i\}_{i=0}^N$. We assume that $a(s)$ is piece-wise constant between two consecutive discretization points, then $b(s)$ becomes a piece-wise linear function. From (3.3.5) we have $2a_i\Delta s_i = b_{i+1} - b_i$ where Δs_i is the size of the ith grid. Now (3.3.6) can be manipulated into

$$T = \int_0^1 \frac{1}{\sqrt{b(s)}} ds = \sum_{i=0}^{N-1} \frac{2\Delta s_i}{\sqrt{b_i} + \sqrt{b_{i+1}}} \tag{3.3.7}$$

A wide variety of constraints, i.e. d in (3.3.1) can be written as $\alpha(s)\dot{s}^2 + \beta(s)\ddot{s} \leq \gamma(s)$ after substituting Eq. (3.3.4) and have to be imposed throughout the trajectory. Due to discretization they are imposed at the mid-points between two consecutive collocation points, i.e. $\alpha(s_{i+1/2})b_{i+1/2} + \beta(s_{i+1/2})a_{i+1/2} \leq \gamma(s_{i+1/2})$. These constraints can be all expressed as linear constraints of b_i, b_{i+1}, after algebraic manipulation.

It turns out all constraints under consideration are linear in b and are denoted as $G_b(p)b \leq g_b(p)$ and $H_b(p)b = h_b(p)$. Note we explicitly write the dependencies of the constraints in the inner problem on the path p. With objective function (3.3.7) being convex, TOPP can be solved by convex optimization. As a result, given a geometric path, we have a reliable

approach to solve the corresponding TOPP.

3.3.4 Bilevel Optimization

The general formulation of bilevel optimization is given Definition 2.1 of Chapter 2. Applying this framework to the time-optimal problem with the parameterization discussed above, the problem is formulated as

$$\begin{aligned} \underset{p,b}{\text{minimize}} \quad & J(p,b) = \textstyle\sum_{i=0}^{N-1} \frac{2\Delta s_i}{\sqrt{b_i}+\sqrt{b_{i+1}}} c^T P(y) \\ \text{subject to} \quad & b \in \underset{b}{\text{argmin}} \{ J(p,b) : G_b(p) b \leq g_b(p);\ H_b(p) b = h_b(p) \} \\ & G_p p \leq g_p \\ & H_p p = h_p. \end{aligned} \tag{TOPT-BO}$$

The algorithm to solve this bilevel problem is similar. The upper level optimizes the geometric path p while the lower level optimizes the temporal variable b for given p and computes gradient for the upper problem. Compared with the bilevel problem in Chapter 2, the upper problem has more complex constraints and the gradient projection is non-trivial given the large amount of constraints on path p to guarantee collision avoidance and path continuity. Instead SNOPT [37] is directly used to handle them and thanks to the proper handling of linear constraints of SNOPT, feasibility of the upper problem is guaranteed.

3.3.5 Interior-Point Method to Certain Duality Gap

For a convex optimization problem with linear constraints in the form of

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & Gx \leq g \\ & Hx = h \end{aligned} \tag{3.3.8}$$

where $f(x)$ is convex. The Karush-Kuhn-Tucker (KKT) conditions for optimality is

$$\begin{aligned} & \tfrac{\partial f}{\partial x} + G^T \lambda + H^T \nu = 0 \\ & Gx + s = g, Hx = h \\ & \lambda \geq 0, s \geq 0, \lambda \circ s = 0 \end{aligned} \tag{3.3.9}$$

where λ and ν are Lagrangian multipliers, s is slack variable and $\circ$ denote element-wise product. The IPM solves Eq. (3.3.8) using Newton's method with special modification to account for inequality constraints and strict complementary slackness. To progressively move towards optimality, the duality gap $\lambda^T s$ is used to control how far the next iteration is to the optimum. In fact, usually a centering step is used in order to prevent $\lambda \circ s$ from reaching 0 too quickly which slows down convergence.

Our modification of the IPM is to change the right side of the strict complementary slackness condition. Instead of solving to $\lambda \circ s = 0$, we only require IPM to solve to $\lambda \circ s = \mu$ for some value $\mu > 0$. Here, μ is a user-defined parameter for our algorithm. This modification has three positive and one negative effects. First, IPM solves toward $\lambda \circ s = 0$ iteratively from a positive initial value of $\lambda \circ s$, the value of $\lambda \circ s$ gradually converges to 0 from some positive value. As a result, early termination at $\mu > 0$ requires fewer iterations. Second, the solution is not on the boundary of nonnegative orthant and we avoid the difficulty preventing warm start. Third, no inequality constraint is active at termination, so there is no need to worry about the active set switch that causes non-smoothness of the outer gradient.

However, the drawback is that the result of optimization is suboptimal, since the KKT condition is not satisfied. The degree of suboptimality can be tuned by reducing the value of μ. Nevertheless, our experiments suggest that even a small value of μ is beneficial, and the benefits overall are worth sacrificing a small amount of optimality in the inner optimization.

Because the duality gap modifies the KKT condition, the gradient obtained by sensitivity analysis is no longer valid. To obtain the new gradient, we use the equivalence between modified KKT conditions and log barrier method [93, Ch. 11]. The optimum of the barrier-augmented problem

$$\begin{aligned} \underset{x,s}{\text{minimize}} \quad & f(x) - \mu \sum \log s \\ \text{subject to} \quad & Gx + s = g, \\ & Hx = h \end{aligned} \tag{3.3.10}$$

satisfies the KKT condition

$$\begin{aligned} & \tfrac{\partial f}{\partial x} + G^T \lambda + H^T \nu = 0 \\ & -\tfrac{\mu}{s} + \lambda = 0 \\ & Gx + s = g, Hx = h \end{aligned} \tag{3.3.11}$$

where the Lagrangian multipliers are λ and ν. Note that the second equality is equivalent to the duality gap condition (3.3.9), and non-negative constraints $\lambda > 0$ and $s > 0$ are implicitly imposed. Hence, we apply sensitivity analysis to the log barrier problem to obtain analytic gradient. Compared with the one without duality gap, the cost function is different and the inequality constraints nows appear in the cost function. The general formulation

stays the same as in Eq. (2.14).

The algorithm for solving the modified KKT system is basically same with standard IPM algorithm as cvxopt [112] and the only difference is Eq. (17) and Eq. (18b) of [112] are modified to account for duality gap μ. The algorithm returns multipliers λ, μ and slack variables s which are used to compute the cost with barrier as in Eq. (3.3.10) and analytic gradient as in Eq. (2.14). Warm start is implemented by keeping the solution (x, s, λ, ν) from the last problem and using them as the initial guess for the next problem. It is possible that some inappropriate step is taken in outer optimization, and warm start fails to solve the inner optimization problem. In this circumstance, a regular initial guess is used.

3.3.6 Duality Gap Bilevel Optimization Algorithm

Our overall bilevel optimization method is presented in Algorithm 3.1. The algorithm takes an initial guess of the geometric path p_0, the linear constraints G_p, g_p, H_p, h_p on the path, and the duality gap μ. It also maintains a set of inner-optimization warm-start parameters, z_{ws}. In each iteration, the TOPP solver (Gap-TOPP) takes a path p, desired duality gap μ, and optional warm-start guess z as input, and outputs an optimized cost J, Lagrange multipliers λ and time parameterization $\{b_i\}_{i=0}^{N}$. The gradient of the path p is computed in the Get-Gradient subroutine using Lagrange multipliers, and later used to update the path. The outer-level optimization is essentially a nonlinear optimization with linear constraints and the optimality conditions are checked by the optimizer. At the end of the algorithm, we solve the gap-free TOPP problem to get lower cost.

Any gradient-based method can be used as the Take-A-Step function which updates p based on gradient ∇ and possibly its history (in Quasi-Newton approaches). We use off-the-shelf NLP solver SNOPT to perform Take-A-Step function. Even though we are using a nonlinear solver, the constraints in outer-level optimization are linear so feasibility is always guaranteed. In our implementation the gap parameter μ is set to a small value and fixed throughout the algorithm.

3.4 NUMERICAL EXPERIMENT

We evaluate our method on random instances of "forests" using the environment generator of Gao et al. [40]. We generated 101 tests, each with a random environment and randomly sampled feasible start points and goal points. We refer to [73] for a visualization of the environment and statistics on number of segments. In all experiments, computation is performed on a PC running Ubuntu 16.04 with 4.00 GHz CPU and 32 GB memory without

Algorithm 3.1: Bilevel-Solver $(p_0, G_p, g_p, H_p, h_p, \mu)$

$p \leftarrow p_0$
$z_{ws} \leftarrow nil$
for $i \leftarrow 0$ *to max-iterations* **do**
  $J, \{b_i\}_{i=0}^N, \lambda, \nu \leftarrow$ Gap-TOPP(p, μ, z_{ws})
  $z_{ws} \leftarrow (\{b_i\}_{i=0}^N, \lambda, \nu)$
  $\nabla \leftarrow$ Get-Gradient(λ, ν)
  $p \leftarrow$ Take-A-Step$(p, J, \nabla, G_p, g_p, H_p, h_p)$
  if *optimality-conditions-satisfied* **then**
    break
  end
end
$J, \{b_i\}_{i=0}^N \leftarrow$ TOPP(p)
Return $J, p, \{b_i\}_{i=0}^N$
def *Gap-TOPP*(p, μ, z)**:**
  if *z is not nil* `// Warm start`
  then
    $\bar{x}, \bar{\lambda}, \bar{\nu} \leftarrow z$
    if $\bar{x}$ *is dynamically infeasible for* p **then**
      Clear warm start
    end
  end
  $x, \lambda, \nu \leftarrow$ solve Eq. (3.3.11) with guess $\bar{x}, \bar{\lambda}, \bar{\nu}$
  return $J(x)$ (Eq. (3.3.10)), x, λ, ν

parallelization. Implementations are mainly based on Python, but the IPM solver is written in C++ based on open-source cvxopt [112]. The NLP solvers are implemented in compiled languages for efficiency, but they are called using cost and constraint functions written in Python.

In this problem setting, a heuristic-based graph search finds a tentative path from the start to goal. Box-type corridors are generated around the path to obtain a list of safe corridors from start to goal that overlap with neighbors. The trajectory is thus divided into several pieces with each piece constrained in one safe corridor. A demonstration of the box-type corridors is shown in Fig. 3.1. We set the log barrier parameter $\mu = 0.001$ in all examples, and keep it constant throughout optimization. However, after optimization terminates we re-solve with $\mu = 0$ to obtain the actual minimum time. To initialize the optimization, we use a minimum jerk trajectory computed using methods in [73] with or without time allocation refinement. The outer optimization has a iteration limit of 80 to limit computation time.

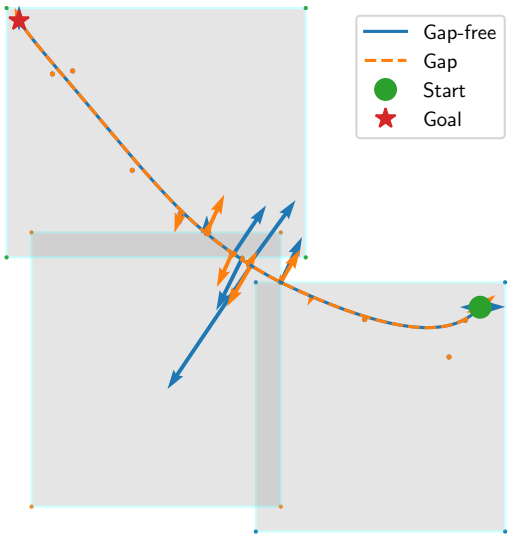


Figure 3.2: Environment of problem A. The blue and orange arrow denotes gradient of control points. The gap-free problem has larger gradient.

3.4.1 Gap vs Gap-free

Here we show the effect of solving our modified KKT system (Gap) by comparing with bilevel optimization where the inner optimization is solved to optimality (Gap free).

Let Problem A be an example composed of 3 boxes shown in Fig. 3.2. The gradient of the cost function with respect to control points is also shown for both solvers. It can be seen that two solvers may have large gradient difference on the same geometric path. Since the gap solver is smoothing the cost function, its gradient has smaller magnitude. An alternate view of this comparison is shown in Fig. 3.3. We start from a path that has already been refined by bilevel solver for a few iterations, with the inner optimizations performed by gap-free solver. Then we compute the gradient direction. We perturb the path slightly by taking small steps along the gradient direction, and plot the directional derivative along the gradient. This figure shows that by introducing the duality gap, a much smoother problem is obtained.

To examine how the duality gap influences the performance of outer optimization, we perform 80 outer iterations on problem A. The summary of results are shown in Tab. 3.1. The gap solver needs fewer number of calls to the inner optimization. On average each inner optimization also requires fewer iteration steps, which is directly proportional to optimization times. As a result, the total computation time is improved. The smoother cost landscape allows more efficient cost improvement given the same amount of outer optimization.

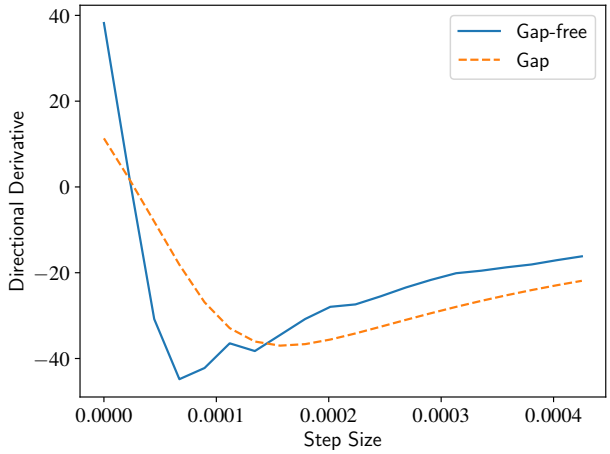


Figure 3.3: Profile of directional derivatives ($\nabla f^* \cdot v/\|v\|$) when a path is moved along a direction v with different step sizes. The gap solver achieves smoother derivatives compared with the gap-free solver.

Table 3.1: Gap vs Gap-free bilevel opt., on problem A

Method	# TOPP	Avg Iter	Comp time (s)	Final cost (s)
Gap	150	8.71	0.37	2.85
Gap-free	168	20.80	0.66	2.89

Table 3.2: Gap vs Gap-free bilevel opt., avg. over all test cases

Method	Comp. Time (s)	Cost Improvement (s)
Gap	0.51	0.39
Gap-free	2.06	0.30

We also compare their difference on all the test cases using the same amount of outer iterations. The results are shown in Tab. 3.2 and are consistent with the results observed in problem A. Since we are solving inner optimization more efficiently, the Gap method achieves lower computation time. The Cost Improvement column refers to the difference between initial traversal time and optimized traversal time, and shows that the smoother cost function obtained by our method leads to faster convergence.

3.4.2 Bilevel vs Nonlinear Optimization

We compare our algorithm with two alternative optimization problem formulations, DC and Joint, both solved with SNOPT and IPOPT. Analytical gradients are provided for solver robustness and sparsity is exploited to the best of the authors' ability.

The Joint condition solves the spatial and temporal parameters jointly. The spatial trajectory is initialized directly using the result from minimum-jerk trajectory since they have the same parameterization. The temporal trajectory is initialized by solving TOPP on the initial guess. With this initialization, the initial guess is always feasible. Both SNOPT and IPOPT are both configured to have optimality tolerance of 10^{-5}. IPOPT has a maximum iteration limit of 200 while SNOPT has total iteration (including both major and minor iterations) limit of 10,000.

The DC condition formulates a nonlinear program using direct collocation. Since DC uses a different trajectory parameterization, we use interpolation on the minimum-jerk trajectory for the initial guess. The initial duration of the trajectory is scaled to satisfy velocity and acceleration bound constraints. When solved by IPOPT, the iteration limit is set as 1,000 and optimality tolerance is 10^{-3}. For SNOPT the total iteration limit is set as 50,000 and the optimality tolerance of 10^{-3} is used.

Success Rate Success rate comparisons are shown in Tab. 3.3. Rows SN and IP denotes the use of the NLP solver SNOPT and IPOPT, respectively, and columns No Ref and Ref denotes whether time allocation is used to refine the minimum jerk trajectory using methods in [73].

Table 3.3: Success rate of planning methods

	DC		Joint		Bilevel	
	No Ref	Ref	No Ref	Ref	No Ref	Ref
SN	78.2%	88.1%	69.3%	93.1%	100%	100%
IP	85.1%	93.1%	20.8%	23.8%		

Table 3.4: Optimized costs and computation times, by problem set ("Set X" indicates those problems on which X succeeds)

Method	DC	Joint	Bilevel
Avg Cost on All			10.03
Avg Cost on Set DC	10.2		9.98
Avg Cost on Set Joint		10.71	10.35
Avg Comp. Time (s)	6.42	0.62	0.51

The bilevel optimization framework maintains inner optimization feasibility by construction and guarantees a feasible solution at any time as long as the initial guess is feasible. As a result, its success rate is 100% as long as IPM is robust. Both Joint and DC rely on an NLP solver and cannot guarantee a feasible solution. In practice, both Joint and DC may end up at an infeasible solution even if the initial guess is feasible. This indicates the strong nonlinearity of time-optimal problem and NLP solver may fail unless a solution close to optimum is provided. Also, it can be seen that SNOPT is better at Joint optimization and IPOPT excels in DC, and providing a refined minimum-jerk trajectory gives higher success rate.

Cost and Computation Time Tab. 3.4 compares our method in terms of the optimized cost and computation time. For DC and Joint, we restrict our comparison to the variants that use the refined initial guess, IPOPT for DC, and SNOPT for Joint. For DC and Joint, we average only the successful results, since the cost of infeasible solution cannot be sensibly measured. The problems that DC and Joint successfully solve are compared with the cost of Bilevel which tends to achieve lower cost. It turns out Bilevel outperforms DC and Joint optimization in terms of both cost and computation time.

It is unclear why Joint has such a success rate but worse cost function, since successful termination requires a feasible solution that satisfies KKT conditions for optimality. Since joint optimization and bilevel optimization are initialized using the same guess, this might be caused by the NLP solver converging to a local optimum.

3.5 CONCLUSION

We present a bilevel optimization framework to solve time-optimal problems with spatial and temporal constraints. We exploit the convexity of TOPP and linearity of spatial constraints to achieve an any-time and highly robust trajectory optimizer. We establish the equivalence of modified KKT condition and log barrier methods. This helps to achieve a highly efficient IPM with warm start and smoother cost function. Numerical experiments show that trajectory optimization on flying robots are solved reliably, while standard NLP solvers fail to maintain feasibility of the solution, take a longer time, and obtain higher cost solutions.

Future work should address in the relation between optimizing log barrier cost and the original cost. The problem of choosing the duality gap and necessity of adjusting it during optimization also requires further investigation. We are also interested in replacing the Bézier curve representation due to its potential conservativeness. Including other constraints such as motor actuation is also a promising direction to extend its area of applications. We are also interested in optimizing types of cost function in the spatial-temporal decomposition framework.

CHAPTER 4: A DATA-DRIVEN INDIRECT METHOD FOR NONLINEAR OPTIMAL CONTROL

Starting from this chapter, the local continuity of the solution to the parametric optimization problem, called the argmin function, is being exploited. Different from previous chapters where the focus is to accelerate isolated optimization problems, from this chapter parametric planning problems are solved, leading to parametric optimization problems. Parametric optimization problems are more challenging compared with solving particular realizations, but the solution can be approximated from precomputed data. In this chapter, the classical k-Nearest Neighbor (k-NN) model is used to approximate the argmin function and help with providing initial guesses to nonlinear optimizers, leading to the Nearest Neighbor Optimal Control (NNOC) framework. It is tested with indirect methods where the unknown costates have no physical meanings and are more difficult to provide compared with direct methods. Compared with random restart technique, NNOC has shown orders of magnitude improvement in terms of computational time and high chances to get global optima. [1]

[1] This chapter is reproduced from Gao Tang and Kris Hauser, "A data-driven indirect method for nonlinear optimal control". In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). A journal version is published in Astrodynamics 2019 Dec.

4.1 INTRODUCTION

Nonlinear optimal control problems are observed in many engineering applications but are still difficult to solve globally with high confidence despite decades of research. Indirect methods derive the necessary conditions for optimality using costate variables, and convert the optimal control problem into a two-point boundary value problem (TPBVP) [6]. The TPBVP contains unknown costates which are usually solved by nonlinear equation solvers. Although indirect methods can be more efficient than direct methods, they are difficult to apply successfully [12]. The derivation of Euler-Lagrange equations are error-prone and algebraic manipulation to form the TPBVP is not-trivial. Moreover, The optimal control might not be continuous, i.e. bang-bang control for some choice of objective functions, and impose challenges to accurate integration of the Euler-Lagrange equations which again makes convergence more difficult. The main difficulty of indirect methods is the costate variables lack physical meanings so good starting values are difficult to provide. For problems with strong nonlinearity, the convergence domain is so narrow that a large number of initial guesses have to be tried to obtain convergence. Hence, they must be augmented with another approach like random restarts to have any chance of obtaining a global optimum, or convergence at all. Such a shortcoming makes indirect methods impractical for real-time applications with strong nonlinearity.

In recent years, the approach of using precomputed data to initialize nonlinear optimizer has received some attention, with some success in trajectory optimization [113] and global nonlinear optimization [114]. The general idea is that a database of solutions can be precomputed among a parameterized set of optimization problems. For a novel problem, if an example exists in the database whose parameters are sufficiently close to its parameters, then solution to the existing problem should be near the solution to the new one. To our knowledge, this approach has not yet been applied to indirect solution of optimal control problems.

We present NNOC, a data-driven implicit optimal control method that attempts to solve TPBVPs using the k nearest problems in the database to determine initial costate guesses. We also present a new technique that uses sensitivity analysis to approximate how solutions vary with respect to problem parameters, which gives more accurate method for guessing initial costates. In our experiments, we study two vehicle control problems of 4D and 5D, respectively. Experiments study how the technique performs compared to other methods, and demonstrate extremely high success rates as the size of the database grows. The use of our proposed sensitivity analysis technique also reduces solution times by approximately 50%.

4.2 RELATED WORK

There is a lot of literature studying how to improve the convergence of indirect methods. In [12, 13] a homotopic approach is used so we can start from an easier problem and gradually approximate the original problem. In [12] a distribution of initial costate variables is proposed which more or less solve the problem that we do not know the bounds of costate variables. Other techniques include grid search of costate variables [115].

The idea of learning from experience has attracted researchers' interest for decades in robotics field. Besides the literature review of learning from expert trajectories in Chapter 1, some more closely related research are briefly mentioned here. Machine learning techniques have been used in trajectory optimization to predict the outcome of a starting point [116] or to predict good initial trajectories [117] for local optimization. The grasps of novel objects are generated using the experience of grasps [118]. In [119] precomputed examples of optimal trajectory are generalized to enable the optimizer to work in real-time, as applied to a robot ball-catching problem.

Our technique is related to explicit model predictive control (MPC) [120], a technique for linearly constrained linear systems with quadratic costs, that analytically computes the piecewise-linear optimal MPC control function over all initial states. Our approach takes a similar approach to nonlinear optimal control using data.

4.3 TRAJECTORY LEARNING

The difficulty of real time trajectory optimization motivates the application of data-driven methods to learn from precomputed optimal trajectories, called trajectory learning in this thesis. Trajectory learning is essentially applying machine learning methods to the argmin function of parametric trajectory optimization problems in parametric planning tasks. It is promising for its flexibility in task parameterization which makes it applicable to a wide range of problems. In this section, the challenges of applying trajectory learning to physical robots are introduced.

4.3.1 Challenges

The flexibility of trajectory learning comes with several challenges. It is supposed that the task is parameterized by vector p under some distribution and for each realization the trajectory optimization problem has a solution $z(p)$. The first challenge is data collection. The distribution of p has to be assumed which may be non-trivial for some applications.

How to parameterize the problem may be an issue as well since the model usually requires fixed size parameter and output. For instance, how to parameterize a 3D environment with unknown information of obstacle types could be difficult. The density and coverage of data have to be sufficient to guarantee model accuracy over the whole domain. Moreover, solving a single optimization problem is already challenging while now the task is to solve n problems. Fortunately, the dataset is built completely offline and highly parallelizable so computation time is not a serious issue as long as there is a reasonable chance of obtaining the optimal solution using random restart techniques for all parameters. Moreover, techniques in this Chapter help to accelerate data generation as well.

The second challenge is how to use the model. Recalling that the model approximates $z(p)$, for a novel problem the model directly predicts the optimal trajectory. Because of the inevitable model prediction error, one may perform a few correction steps to reduce the violation of dynamics and safety constraints by either solving trajectory optimization using prediction as an initial guess or iterative LQR type corrections. Whatever correction is used, even if no correction, the result is an open-loop trajectory. In order to execute the trajectory on physical systems, it is common to design a feedforward-feedback trajectory tracking controller. The trajectory tracking controller is necessary since the dynamics model used in trajectory optimization may be incorrect due to unmodelled disturbances and the trajectory may not be dynamically feasible due to approximation errors. The whole process of designing and applying a tracking controller for a trajectory and measuring if the goal state is reached (within some threshold) is called *trajectory rollout*. The usual way to evaluate the performance of a machine learning model is to compute the average prediction error on the test set. However, for trajectory learning a more meaningful performance metric is the success rate of trajectory rollout. For problems with strong requirements of constraint satisfaction e.g. collision avoidance, one performance metric is the constraint violation of the predicted trajectory.

The last challenge is model fitting. The goal is to improve the model accuracy by appropriate model selection and hyperparameter tuning. The most difficult part of model fitting is to make sure the model has decent accuracy in the whole domain of the argmin function i.e. support of the distribution of p for safety concerns. Any poorly predicted trajectory that has large violation of system dynamics and safety constraints may lead to task failure and has to be avoided. The reason for the requirement of dynamic feasibility is trajectory rollout depends on the predicted trajectory for both feedforward and feedback controller design. It is important that the feedforward control is not far from the optimum and linearization around the predicted trajectory is valid which requires the predicted trajectory to be close to the actual trajectory. Unfortunately, it is quite difficult to quantify what magnitude of

prediction error is small enough, although the feedback controller allows some margin of error. Nevertheless, it is safe to assume the model must have decent accuracy in the whole support of the distribution of p for a high rollout success rate. Besides requirements of the coverage, amount, and quality of data, the model capability to approximate the complicated argmin function, and appropriate method to train the model are also needed. One overlooked phenomenon–discontinuity of argmin is bringing difficulty to continuous models such as neural networks. The neural network tends to predict an "average" of significantly different trajectories near discontinuity regions. The prediction is far from all the neighboring trajectories and leads to large prediction error and eventually task failure. It is thus important to study how to learn a discontinuous function.

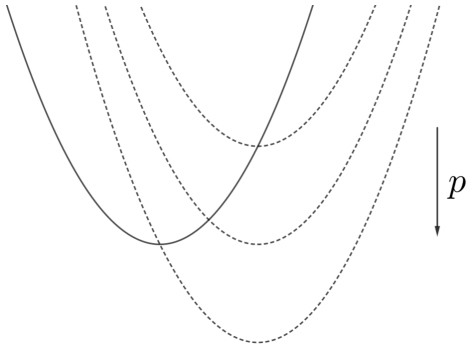
4.3.2 Discontinuous Function Learning

Discontinuity in the argmin function means problems with similar parameters have quite different optimal trajectories. For a general parametric optimization problem, this may occur due to switch of local optimal family, ambiguity of the solution (when the optimal solution is a set), and some degeneration of problems. The first example as shown in Fig. 4.1a is

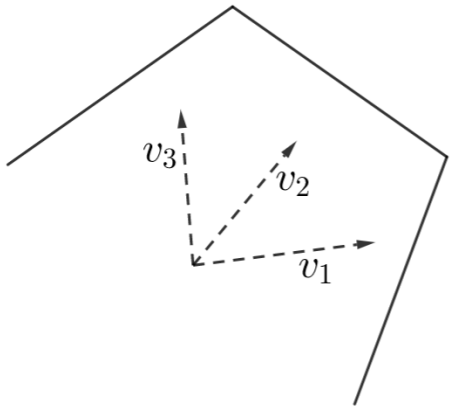
$$\underset{x}{\text{minimize}} \quad \min(x^2, (x-1)^2 + p) \tag{4.3.1}$$

whose argmin is $x^* = 0$ for $p \geq 0$ and $x^* = 1$ for $p < 0$. The optimal solution has discontinuity as p changes between being positive and negative. Another example is shown in Fig. 4.1b with objective function to maximize $v^T x$ subject to linear constraints. As v changes from v_1 to v_3, the optimum switches from right corner to top corner since at v_2 the optimum is ambiguous. This examples also shows the argmin may still have discontinuity even for parametric convex problems.

Applying trajectory learning in many robotic applications faces the discontinuous argmin problem as well. As shown in later chapters, the discontinuity may come from switch of trajectory homotopy class and ambiguity caused by geometric singularity. It is thus important to consider those discontinuities in order to guarantee the model performance even in regions near discontinuity. The k-NN model is intrinsically discontinuous and requires no interpolation so it can handle discontinuous function. In this chapter, k-NN is used to approximate the argmin and provide initial guess for nonlinear optimizers. remove me comment



(a) Discontinuity from switch of local optima



(b) Discontinuity from solution ambiguity

4.4 DATA-DRIVEN FRAMEWORK

4.4.1 Indirect Methods for Optimal Control

Indirect methods essentially convert an optimal control problem to a system of nonlinear equations. Suppose a nonlinear optimal control problem is given as

$$\begin{aligned} &\text{Minimize} && J = \varphi(t_0, x_0, t_f, x_f) + \int_{t_0}^{t_f} \ell(t, x, u)\mathrm{d}t \\ &\text{subject to} && \dot{x} = f(t, x) \\ & && c(u) \leq 0 \\ & && h(t_0, x_0, t_f, x_f) = 0 \end{aligned} \tag{4.4.1}$$

where $x \in \mathbb{R}^n$ is the state variable, $u \in \mathbb{R}^m$ is the control, and $t \in [t_0, t_f]$ is time. The path constraint c collects inequality constraints on control. We note that indirect methods have difficulty handling path constraint of states so they are assumed not to exist. h is a collection of equality constraint on state variables at initial and final time. For simplicity we will ignore other types of constraints. The indirect method introduces corresponding costate variables $\lambda \in \mathbb{R}^n$ and the Hamiltonian [6]

$$H = L(t, x, u) + \lambda^{\mathrm{T}} \dot{x} = L(t, x, u) + \lambda^{\mathrm{T}} f(t, x, u) \tag{4.4.2}$$

and derive the Euler-Lagrange equations

$$\begin{cases} \dot{x} = \dfrac{\partial H}{\partial \lambda} \\ \dot{\lambda} = -\dfrac{\partial H}{\partial x} \end{cases} \tag{4.4.3}$$

and the optimal control

$$u^\star = \underset{c(u)\leq 0}{\operatorname{argmin}} H \tag{4.4.4}$$

It is from this latter condition that the optimal control $u^\star$ as a function of x and λ can (usually) be determined. If the control u has no constraint, we uses $\partial H/\partial u = 0$ to calculate $u^\star$. For example, suppose

$$L(t, x, u) = x^{\mathrm{T}} Q x + u^{\mathrm{T}} R u$$

be a quadratic cost and

$$f(t, x, u) = f_0(t, x) + \sum_{i=1}^{m} f_i(t, x) u_i$$

be the dynamics function. Then

$$\frac{\partial H}{\partial u} = 2Ru + \begin{bmatrix} \lambda^{\mathrm{T}} f_1(t, x) \\ \vdots \\ \lambda^{\mathrm{T}} f_m(t, x) \end{bmatrix} = 0 \tag{4.4.5}$$

and hence u can be determined by multiplying the second summand by $-R^{-1}/2$.

Then, to solve for the optimal trajectory given two-point boundary conditions including an initial state,

$$x(t_0) = x_0 \tag{4.4.6}$$

and final state

$$x(t_f) = x_f, \tag{4.4.7}$$

a shooting method is used to determine the unknowns so that the boundary condition at t_f in Eq. (4.4.7) is satisfied. For the fixed-time problem where t_0 and t_f are fixed, the unknowns z are the initial costate variables $\lambda(t_0)$.

In this chapter we also consider problems with free t_f. In those problems the final time t_f is also a unknown, so $z \equiv (\lambda, t_f)$ and another boundary condition at t_f will be imposed. Specifically, the following statistic condition should be imposed:

$$H(t_f) = 0. \tag{4.4.8}$$

The TPBVP solver guesses all the unknowns z and integrates Eqs. (4.4.3) simultaneously using the optimal control from time t_0 to t_f. The unknowns are updated until the boundary conditions (i.e. Eq. (4.4.7) for fixed t_f) at t_f are satisfied up to a given tolerance. We implement the TPBVP shooting method by using the nonlinear least-squares software Minpack

[121]. Note that these least-squares solvers, typically based on Gauss-Newton or Levenberg-Marquardt methods, are only local optimizations and may indeed fail to find a solution that successfully meets the final state. An initial guess close to the solution is required otherwise convergence cannot be guaranteed. In practice random restart method is usually employed to find the global optimal solution. Thus the efficiency and reliability of indirect methods are limited.

4.4.2 Parametric Optimal Control with Varying Initial Conditions

NNOC addresses the span of optimal control problems ranging over all initial conditions, but with a fixed final state $x_f = 0$. In the parametric trajectory optimization framework, the planning problem is parameterized by the initial state, i.e. $p \equiv x_0$. A complete, globally optimal method for solving the optimal control problem can be viewed as a mapping g from x_0 to the optimal solution of the unknowns:

$$g : x_0 \to z^\star. \tag{4.4.9}$$

The goal of NNOC is to approximate this map. Assuming x_0 is defined in set X, the first step to build the database is sampling from X and calculating the corresponding global optimal solution. We can thus form a database of parameter-solution pairs where x_0 is the parameter and $z_0^\star$ is the solution. Specifically, the database $D = \{(x_0^{(i)}, z_0^{\star(i)}) | i = 1, \ldots, N\}$ where we denote $(x_0^{(i)}, z_0^{\star(i)})$ as an optimal control pair. Since the computation of the database is offline, we employ heuristic global optimization techniques such as random restarts. It should be noted that for a general nonlinear optimal control problem there is no guarantee that a global optimal solution can be found. The best local optimal solution is considered as the global optimal solution so a sufficiently large number of restarts is used. However, it is possible that for certain initial states no solution exist or we fail to find a feasible solution. In that case we simply mark that no solution exists.

4.4.3 Extending the Database along Trajectories

We observe that each successful trajectory solve provides not only the optimal costate at the initial time t_0, but also all times thereafter. Hence, we can populate the database more quickly by generating optimal problem/solution pairs $(x(t), z(t))$ along the trajectory. So, after calculating an optimal trajectory $x(t)$, costate trajectory $\lambda(t)$, and optionally the final time t_f, we evenly sample M states and costates along the trajectory and add their

examples to the database. Specifically, we divide the time range $[t_0, t_f]$ at intermediate values $t_i, i = 1, \ldots, M$, and add all of $(x(t_i), z(t_i))$ to the database. In the case where z contains final time we set the final time for point i to be $t_0 + t_f - t_i$.

4.4.4 Query of Database

For a novel problem, NNOC performs a k-nearest neighbor query of the database, and the costates for each of the k nearest neighbors are used as seeds for the TPBVP solver. The feasible solution with minimum cost is kept. The NN query is performed using the approximate nearest-neighbors library FLANN [122] [2]. Several parameters affect performance, including:

1. Database size N. If N is too small, the distance between new parameters and its nearest neighbors might be too large, so the solutions might not lead to convergence. Larger N is needed for problems that are highly nonlinear with small convergence domain. Nearest-neighbor lookup time is fairly insensitive to N due to the use of approximate methods.

2. Distribution of the examples in the database should approximate the query distribution. A naïve method is to sample each state component uniformly at random in a range, but if knowledge is available about which states are encountered in practice, it should be employed.

3. Number of neighbors k determines how many precomputed solutions are used as tentative values for new problems. A larger k contributes to the robustness by combating the effects of local optimal solutions. However, larger values increase running time.

We also present the option to employ sensitivity analysis when determining an initial guess to a novel problem. Rather than using precomputed solutions directly as the initial guess for new problems, this method builds a first-order approximation of g to determine a better guess. It is described below.

4.4.5 Sensitivity Analysis

Assuming the mapping from parameters to solutions is continuous and differentiable, sensitivity analysis can be used to build a first-order approximation to their variations.

[2] http://www.cs.ubc.ca/research/flann/ (accessed 9/2/2017)

We can thus obtain a better initial guess than directly using the solutions of precomputed problems.

NNOC queries the database to find neighbors of the new state. But instead of using the solutions of the neighbors directly as the initial guess, we can obtain a better guess using the first-order approximation of the relation between the variation of parameters and the variation of solutions.

Let us first explain the method for the fixed-time case. Denote $\lambda_0 \equiv \lambda(t_0)$. We take the variation of (4.4.7) and obtain

$$\frac{\partial x(t_f)}{\partial x_0}\delta x_0 + \frac{\partial x(t_f)}{\partial \lambda_0}\delta \lambda_0 = 0 \tag{4.4.10}$$

where $\frac{\partial x(t_f)}{\partial x_0}$ and $\frac{\partial x(t_f)}{\partial \lambda_0}$ are easily obtained by integrating the variational equation of the system dynamics. We refer to [123] for further details.

Using (4.4.10) we can obtain a linear relationship between the change of initial state and the change of initial costate variables

$$\delta \lambda_0 = \frac{\partial \lambda_0}{\partial x_0}\delta x_0 = -\frac{\partial x(t_f)}{\partial \lambda_0}^{-1}\frac{\partial x(t_f)}{\partial x_0}\delta x_0. \tag{4.4.11}$$

It should be noted that the matrix $\frac{\partial \lambda_0}{\partial x_0}$ can be computed offline since it is determined by x_0 and λ_0 and can be computed when the database is being built.

Then, when a query problem x_0 is matched to an example (x, λ) in the database (noting $\lambda = z$ in this example), we seed the solver with the initial costate

$$\lambda + \frac{\partial \lambda}{\partial x}(x_0 - x). \tag{4.4.12}$$

For free-time problems, we must compute the sensitivity of both λ_0 and t_f with respect to x_0. To do so, compute the variation of (4.4.7) and (4.4.8), obtaining

$$\begin{cases} \frac{\partial x(t_f)}{\partial x_0}\delta x_0 + \frac{\partial x(t_f)}{\partial \lambda_0}\delta \lambda_0 + \frac{\partial x(t_f)}{\partial t_f}\delta t_f = 0 \\ \frac{\partial H(t_f)}{\partial x_0}\delta x_0 + \frac{\partial H(t_f)}{\partial \lambda_0}\delta \lambda_0 + \frac{\partial H(t_f)}{\partial t_f}\delta t_f = 0 \end{cases} \tag{4.4.13}$$

Here, $\frac{\partial x(t_f)}{\partial x_0}$ and $\frac{\partial x(t_f)}{\partial \lambda_0}$ are obtained as before; $\frac{\partial x(t_f)}{\partial t_f} = \dot{x}(t_f)$ is obtained by substituting the optimal control into the dynamics function; $\frac{\partial H(t_f)}{\partial t_f} = 0$ since H does not depend on

time [6]; $\frac{\partial H(t_f)}{\partial x_0} = \frac{\partial H(t_0)}{\partial x_0} = -\dot{\lambda}(t_0)$; and $\frac{\partial H(t_f)}{\partial \lambda_0} = \frac{\partial H(t_0)}{\partial \lambda_0} = \dot{x}(t_0)$. Then we can calculate $\delta\lambda_0$ and δt_f by solving a system of $n+1$ linear equations.

$$\begin{bmatrix} \delta\lambda_0 \\ \delta t_f \end{bmatrix} = - \begin{bmatrix} \frac{\partial x(t_f)}{\partial \lambda_0} & \frac{\partial x(t_f)}{\partial t_f} \\ \frac{\partial H(t_f)}{\partial \lambda_0} & \frac{\partial H(t_f)}{\partial t_f} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial x(t_f)}{\partial x_0} \\ \frac{\partial H(t_f)}{\partial x_0} \end{bmatrix} \delta x_0. \tag{4.4.14}$$

As with the fixed-time case, the sensitivity matrix can be done offline and stored with the example in the database.

4.5 RESULT

We test NNOC in two problems. Planar Car is a minimum effort problem on a second-order Dubins car. Quadcopter is a minimum-time problem on a dynamic quadcopter model. In each case we seed the database using randomly sampled initial states, solved using a random restart method. The first example is a basic demonstration of the data-driven technique. Test sets of 1,000 problems are randomly generated from the same distributions as we generate the training set. The database was extended in the Quadcopter case with $M = 100$ trajectory samples. This example demonstrates we can use the technique of extending database along trajectories to enlarge our database. In order to use the database, the states of the quadcopter when it is moving towards the target are also added to the test set. We study the effects of parameters such as the stopping criteria for random restart, database size, number of neighbors queried, and whether sensitivity analysis is used. The Planar Car experiments are run on a Macbook Pro with a 2.90 GHz CPU while the Quadcopter experiments are run on a desktop with a 3.60 GHz CPU.

4.5.1 Planar Car

We consider a simplified planar car with the following dynamics [124]:

$$\dot{x} = v\sin\theta,\ \dot{y} = v\cos\theta,\ \dot{\theta} = u_\theta v,\ \dot{v} = u_v \tag{4.5.1}$$

where the state $x = [x, y, \theta, v]$ includes the planar coordinates, orientation, and velocity of the vehicle. The control $u = [u_\theta, u_v]$ includes the control variables which change the steering angle and velocity, respectively.

Optimal Control Formulation The performance index is given by the quadratic control effort

$$J = \int_{t_0}^{t_f} u^{\mathrm{T}} R u \mathrm{d}t \tag{4.5.2}$$

where R is a diagonal matrix with entries $r_1 = 0.2$ and $r_2 = 0.1$, as chosen in accordance with [124].

We arbitrarily choose fixed initial and final times, i.e. $t_0 = 0, t_f = 2.5$. Initial states are sampled from the range $x \in [-3, 0], y \in [-3, 3], \theta \in [-\pi, \pi], v = 0$. It should be noted that due to the symmetry of the problem we do not have to consider positive x. The target state is the origin, so that final orientation and velocity is 0.

The costate variables $\lambda = [\lambda_x, \lambda_y, \lambda_\theta, \lambda_v]$ are governed by the Hamiltonian [6]

$$\begin{aligned} H &= \lambda^{\mathrm{T}} \dot{x} + u^T R u \\ &= \lambda_x v \sin\theta + \lambda_y v \cos\theta + \lambda_\theta v u_\theta + \lambda_v u_v + r_1 u_\theta^2 + r_2 u_v^2 \end{aligned} \tag{4.5.3}$$

and we derive the Euler-Lagrange equations

$$\begin{cases} \dot{\lambda}_x = -\dfrac{\partial H}{\partial x} = 0 \\ \dot{\lambda}_y = -\dfrac{\partial H}{\partial y} = 0 \\ \dot{\lambda}_\theta = -\dfrac{\partial H}{\partial \theta} = -\lambda_x v \cos\theta + \lambda_y v \sin\theta \\ \dot{\lambda}_v = -\dfrac{\partial H}{\partial v} = -\lambda_x \sin\theta - \lambda_y \cos\theta - \lambda_\theta u_\theta \end{cases} \tag{4.5.4}$$

Then we readily derive the optimal control which minimizes H as

$$u_\theta^* = -\frac{v\lambda_\theta}{2r_1},\ u_v^* = -\frac{\lambda_v}{2r_2} \tag{4.5.5}$$

Estimation of magnitude of costate variables The difficulty for providing tentative $\lambda(t_0)$ is partially caused by the its unknown bounds. Admittedly, with the help of the normalization of initial costate variables [12] we can sample them on the surface of a high-dimension ball. Here we use a non-rigorous formula to help provide bounds for initial costate variables. Experiments show that it helps to increase success rate of random restart technique. The formula we use for estimation of the magnitude of λ_v is

$$\bar{s} = 2\sqrt{x^2 + y^2}, \bar{v} = \frac{\bar{s}}{t}, \bar{a} = \frac{4\bar{v}}{t}, \bar{\lambda}_v = 2\bar{a} r_2 \tag{4.5.6}$$

where $\bar{(\cdot)}$ denotes the magnitude. We first estimate the length of the trajectory, then average velocity and average acceleration assuming a constant acceleration and deceleration. Then the formulation of the optimal control is used to get the magnitude of λ_v. Using a similar way, we obtain the magnitude of λ_θ as

$$\bar{\lambda}_\theta = \frac{16|\theta_0|r_1}{t^2\bar{v}^2} \tag{4.5.7}$$

It should be noted that an additional term $\bar{v}^2$ is multiplied in the denominator because the additional multiplication of v in Eqs. (4.5.5) and (4.5.1).

Simulation result We evaluate four methods that differ in how initial guesses are provided and how the iteration is terminated.

1. RR: Random Restart of k initial guesses, where $k = 5, 10, 50, 100$
2. RL: Random restart after k Locally optimal solutions are solved, where $k = 1, 3, 5, 10$
3. NNOC: start with the solutions of the k Nearest Neighbors where $k = 1, 5, 10$
4. NNOC+SA: start with the solutions of the k Nearest Neighbors where $k = 1, 5, 10$ with Sensitivity Analysis

It should be noted that in RL a maximum restart number of 100 is set in order to avoid an infinite loop. For random restart techniques, the normalization of initial costates is used [12].

To build the database, we randomly generate 20,000 initial states and calculate the corresponding costates. Then 5 databases of size 1,250, 2,500, 5,000, 10,000, 20,000 are constructed, respectively. In Fig. 4.2 we plot the optimal trajectories of 30 examples where the arrow shows the direction the car is heading. We record the running time, number of solutions, and the best performance index of each method. The results for RR and RL are listed in Tab. 4.1 and Tab. 4.2, respectively. Success Rate denotes the percentage of obtaining at least one local optimal solution; Avg. Conv. denotes the average number of local optima; Global Rate denotes the fraction of times a method obtains a globally optimal solution; Avg. Time is the average computation time for solving the TPBVP using shooting methods. To calculate Global Rate, for each test example we compute the minimum cost solution found over all methods tested. This value is then considered the globally minimum cost. A local optimum returned by each method is considered global if its cost is close to the minimum cost (the difference is less than 1×10^{-6}). For this problem, we find that for a desired Global

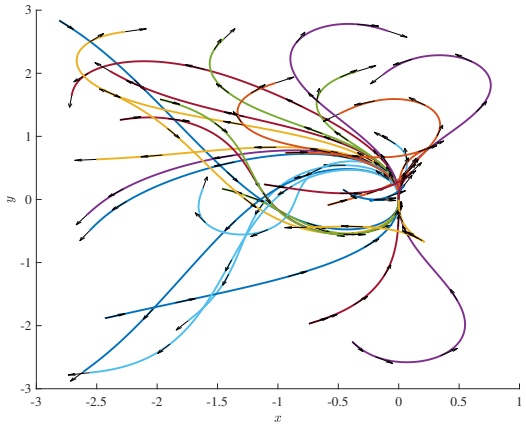


Figure 4.2: Sample of optimal trajectories for Planar Car example.

Optimal Rate of 95%, about 100 random initial guesses must be tried, and RR takes about 7 s.

Table 4.1: Results of RR for Planar Car Example

k	Success Rate	Avg. Conv.	Global Rate	Avg. Time (s)
5	0.673	1.145	0.494	0.378
10	0.862	2.385	0.684	0.769
50	0.997	11.824	0.939	3.831
100	1	23.697	0.966	7.649

Fig. 4.3 compares running times to NNOC, and Fig. 4.4 shows the Global Optimal Rate, for differing numbers of database size N, neighbor count k, and whether sensitivity analysis is enabled. It can be observed that the average computation time decreases with increasing N. The influence of N on average computation time is more significant for larger k. This is reasonable since when k is large, the distance of some neighbors might be quite large and increasing N can help avoid initial guesses that takes a long time to converge. Increasing k increases global optimal rate significantly, but at the cost of increasing computation time. Sensitivity analysis significantly reduces the average computation time and increases global optimal rate. For a target global optimal rate of 95%, $k = 5$ and $N = 20,000$ leads to average computation time of 0.05 s which is nearly two orders of magnitude faster than RR.

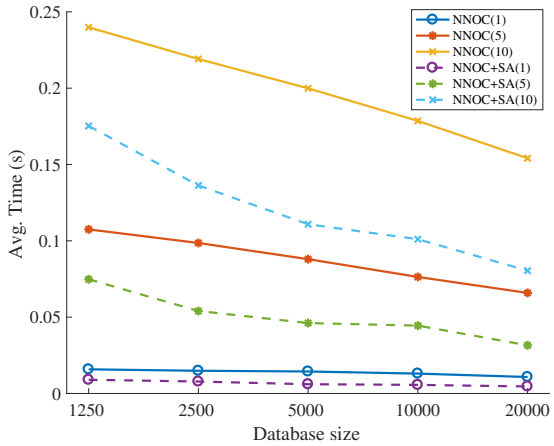


Figure 4.3: Average computation time of NNOC for Planar Car example.

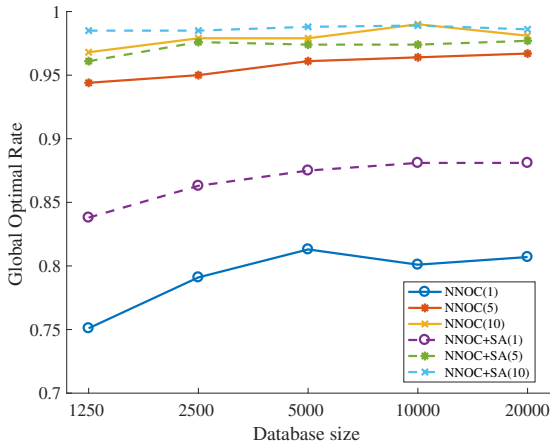


Figure 4.4: Global Optimal Rate of NNOC for Planar Car example.

Table 4.2: Results of RL for Planar Car Example

k	Success Rate	Avg. Conv.	Global Rate	Avg. Time (s)
1	1	5.679	0.610	0.562
3	0.992	15.516	0.854	1.285
5	0.982	25.070	0.910	1.982
10	0.932	45.002	0.917	3.276

4.5.2 Planar Quadcopter

The Quadcopter example defines the following dynamics [125]:

$$\ddot{x} = \frac{F_T}{m}\sin\theta,\ \ddot{z} = \frac{F_T}{m}\cos\theta - g,\ \dot{\theta} = \omega \tag{4.5.8}$$

where g is the gravitational acceleration; m the mass of the quadcopter; F_T the total thrust force; and ω the pitch rate. The state $x = [x, z, \dot{x}, \dot{z}, \theta]$ include the x, z coordinates, the velocity, and the pitch angle. The control is defined as $u = [u, \omega]$ where $u = F_T/m$. Both controls are subject to saturation:

$$\underline{u} \leq u \leq \overline{u},\ |\omega| \leq \overline{\omega} \tag{4.5.9}$$

where $\underline{u} = 1, \overline{u} = 20, \overline{\omega} = 5$. The initial state of the quadcopter is randomly sampled such that $x \in [-10, 0], z \in [-10, 10], \dot{x} = \dot{z} = \theta = 0$. Due to symmetry of the problem, we do not have to sample the cases with positive x.

Optimal Control Formulation The objective in this problem is to move to and hover at the origin in minimum time. The time-optimal performance index is

$$J = \int_0^{t_f} 1 \mathrm{d}t \tag{4.5.10}$$

where t_f is a free variable. However, the resulting optimal control is bang-bang control which is numerically challenging to solve [125]. Hence, we use an alternate formulation that adds regularization parameters to the performance index so the resulting optimal control is continuous [126]. We introduce a logarithmic barrier function to the performance index,

which is a widely-used method in the field of aerospace engineering. [13, 15], as follows,

$$\begin{aligned} J &= \int_0^{t_f} L \, \mathrm{d}t \\ &= \int_0^{t_f} 1 - \epsilon_1 \ln[\hat{u}(1-\hat{u})] - \epsilon_2 \ln[\hat{\omega}(1-\hat{\omega})] \, \mathrm{d}t. \end{aligned} \tag{4.5.11}$$

where $\hat{u} \in [0,1]$ and $\hat{\omega} \in [0,1]$ are nondimensionalized controls such that $u \equiv \underline{u} + (\overline{u} - \underline{u})\hat{u}$ and $\omega \equiv \underline{\omega} + (\overline{\omega} - \underline{\omega})\hat{\omega}$. It can be shown that with this modification, the resulting optimal control is continuous and differentiable. The parameters ϵ_1 and ϵ_2 control the magnitude of the logarithmic barrier. Smaller values lead to a better approximation to the bang-bang control, but they also enlarge numerical sensitivity and thus reduce the convergence domain. In this work we initially choose the values $\epsilon_1 = \epsilon_2 = 1$ which are relatively large compared with [15], but also have a larger convergence domain as the optimal control problem becomes less ill-conditioned.

With costate variables $\lambda = [\lambda_x, \lambda_z, \lambda_{\dot{x}}, \lambda_{\dot{z}}, \lambda_\theta]$, we write the Hamiltonian as

$$H = \lambda_x \dot{x} + \lambda_z \dot{z} + \lambda_{\dot{x}} u \sin\theta + \lambda_{\dot{z}}(u\cos\theta - g) + \lambda_\theta \omega + L \tag{4.5.12}$$

and derive the Euler-Lagrange equations

$$\begin{cases} \dot{\lambda}_x = -\dfrac{\partial H}{\partial x} = 0, \; \dot{\lambda}_y = -\dfrac{\partial H}{\partial x} = 0 \\ \dot{\lambda}_{\dot{x}} = -\dfrac{\partial H}{\partial \dot{x}} = -\lambda_x, \; \dot{\lambda}_{\dot{z}} = -\dfrac{\partial H}{\partial \dot{z}} = -\lambda_z \\ \dot{\lambda}_\theta = -\dfrac{\partial H}{\partial \theta} = \lambda_{\dot{z}} u \sin\theta - \lambda_{\dot{x}} u \cos\theta \end{cases} \tag{4.5.13}$$

Then the non-dimensionalized optimal control that minimizes H is

$$\begin{cases} u^* = \dfrac{2\epsilon_1}{2\epsilon_1 + \rho_1 + \sqrt{4\epsilon_1^2 + \rho_1^2}} \\ \omega^* = \dfrac{2\epsilon_2}{2\epsilon_2 + \rho_2 + \sqrt{4\epsilon_2^2 + \rho_2^2}} \end{cases} \tag{4.5.14}$$

where ρ_1 and ρ_2 are called switching functions and are defined as

$$\rho_1 = (\overline{u} - \underline{u})(\lambda_{\dot{x}} \sin\theta + \lambda_{\dot{z}} \cos\theta), \; \rho_2 = (\overline{\omega} - \underline{\omega})\lambda_\theta. \tag{4.5.15}$$

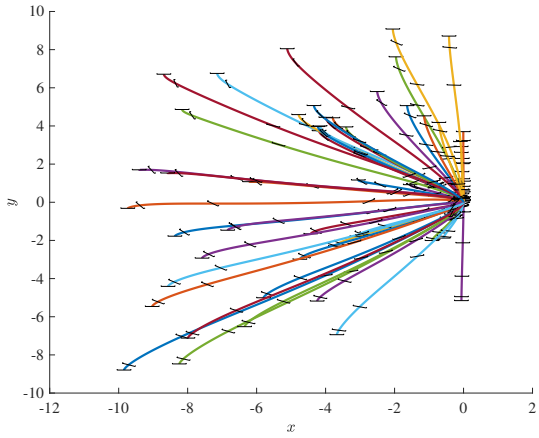


Figure 4.5: Sample optimal trajectories for the Quadcopter example.

Simulation Result In this example, a different approach is applied to compare RR and data-driven technique. We use the same method as we generate the training set to get 500 samples for the test set. We use the same way of extending the database along trajectories to generate testing examples. Along each trajectory we evenly sample $l = 10$ states. The following results is a summary of the simulation results on the sampled initial states.

To build the database, we randomly generate 2,000 initial states and calculate the corresponding costates. Then the technique of sampling optimal trajectories to extend the database is applied. 5 databases of size 12,500, 25,000, 50,000, 100,000, 200,000 are constructed, respectively. In Fig. 4.5 we plot the optimal trajectories of 50 examples. The results for RR and RL are listed in Tab. 4.3- 4.4. We note that for this problem getting local optima is not a serious problem and the Global Optimal Rate approximately equals the success rate of returning one result. Random restart until 1 local optimal solution is found is a good choice to solve this problem. On average this takes about 0.2 s.

Table 4.3: Results of RR for Planar Quadcopter Example

k	Success Rate	Avg. Conv.	Global Rate	Avg. Time (s)
5	0.732	2.030	0.732	0.244
10	0.799	4.063	0.899	0.490
50	0.853	20.137	0.853	2.448
100	0.861	40.285	0.861	4.901

Fig. 4.6 shows solving time for NNOC. Increasing k leads to longer solving time since more instances of TPBVP must be solved. But increasing the database size and using sensitivity

Table 4.4: Results of RL for Planar Quadcopter Example

k	Success Rate	Avg. Conv.	Global Rate	Avg. Time (s)
1	0.861	4.111	0.861	0.191
3	0.848	10.181	0.861	0.542
5	0.832	15.150	0.861	0.850
10	0.789	25.376	0.861	1.494

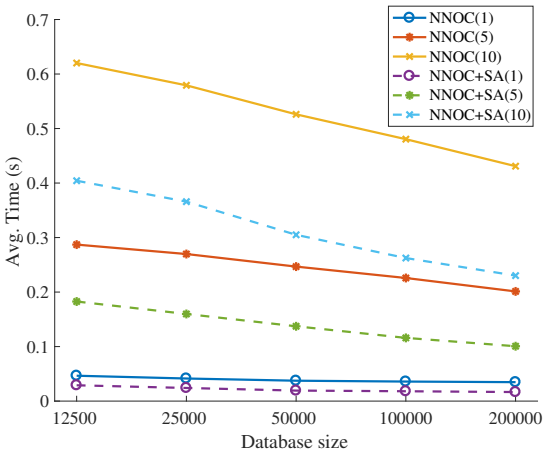


Figure 4.6: Average computation time of NNOC for Quadcopter example

analysis decreases computation time. However, when $k = 1$, the decrease in computation time is quite small. Fig. 4.7 shows the Global Optimal Rate, showing that NNOC almost always computes the global optimum. When $k = 1$ and sensitivity analysis, the global optimum is found about 99% of the time and average computation time is less than 4 ms.

Model Predictive Control Simulation A potential application of NNOC is to implement real-time nonlinear MPC, which we illustrate using the simulated quadcopter problem under disturbances (e.g., wind gusts).

We simulate model uncertainty by adding small perturbation to the system dynamics. The simulated Eq.(4.5.8) becomes

$$\ddot{x} = \frac{F_T}{m}\sin\theta + a_x,\ \ddot{z} = \frac{F_T}{m}\cos\theta - g + a_z,\ \dot{\theta} = \omega + \alpha\Delta t \tag{4.5.16}$$

where a_x and a_z are the unmodelled acceleration along x and z axes; α is the unmodelled angular acceleration; Δt is the instantaneous time from the last control step such that $\alpha\Delta t$

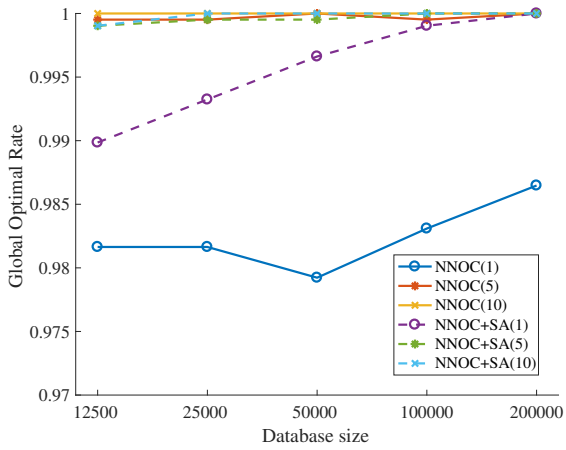


Figure 4.7: Global Optimal Rate of NNOC for Quadcopter example

is the unmodelled change of angular velocity. At every control step ($\Delta t = 0.1$), a_x, a_z, α are sampled from normal distributions with standard deviations of 1.

Our implementation of the NNOC-MPC controller receives the current state as input at 10Hz and uses NNOC+SA with $k = 5$ nearest neighbors to solve the optimal control problem. In some cases, convergence cannot be obtained using NNOC. In this case, we try solving using the costate trajectory from the last step as the initial guess, integrating Eq.(4.5.13) forward by Δt. If convergence cannot be reached again, then we consider this a failure of the controller. The controller repeats until 1) the quadcopter is close to the origin, as measured by $\|x\| < 2$ or 2) the quadcopter is too far from the origin, as measured by $\|x\| > 20$.

Our first NNOC database simply uses states along the optimal trajectories from states at rest, as before. However, this database does not achieve adequate coverage in areas of the state space that the system might reach under disturbances. We consider generating a *robust database* by simulating the NNOC-MPC controller from random initial states using a larger perturbation and collecting the state and solutions. If the novel state is not solved successfully using the above technique, a brute-force RR(100) is used. Each database has 500,000 parameter-solution pairs. Symmetry about the x axis is exploited by sampling states only with $x \leq 0$, and mirroring any states with $x > 0$. Specifically, if $(x, z, \dot{x}, \dot{z}, \theta)$ and $(\lambda_x, \lambda_z, \lambda_{\dot{x}}, \lambda_{\dot{z}}, \lambda_\theta)$ form a state-solution pair, then $(-x, -z, \dot{x}, \dot{z}, -\theta)$ and $(-\lambda_x, \lambda_z, -\lambda_{\dot{x}}, \lambda_{\dot{z}}, -\lambda_\theta)$ also form a pair.

We perform 500 simulations with random initial states as shown in Fig. 4.8. We record failed NNOC solves (which can be corrected during execution using a backup solver), as well as overall failures of convergence. Table 4.5 gives a numerical comparison. Using the standard database, the controller fails to achieve solutions via NNOC in 1.6% of intermediate

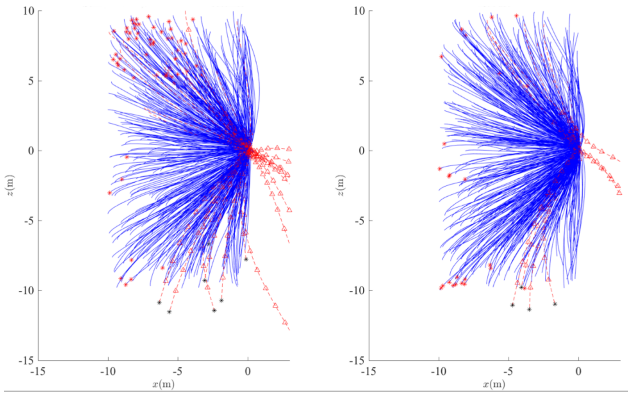


Figure 4.8: Comparison of NNOC-MPC simulation on the quadcopter for two databases. Left: the standard database is constructed by sampling along unperturbed optimal trajectories. Right: the database is generated by running MPC under perturbation. Blue trajectories are successful, and red are unsuccessful. Red stars and triangles denote states that are not solved by NNOC-MPC. Black stars mark the final states of failed trajectories (some are not visible in the depicted range).

Table 4.5: Results for MPC with NNOC on the Quadcopter problem

	Standard DB	Robust DB
NNOC+SA solve failures	1.6%	**0.65%**
Convergence failures	3.2%	**1.4%**

states. This also leads to overall failure of convergence in 16 / 500 initial states. Using the robust database, NNOC+SA failed to obtain solutions in 0.65% of intermediate optimal control problems. This leads to a failure of convergence in 7 / 500 initial states. These results demonstrate that NNOC is able to control model uncertainty in a large fraction of trajectories, and that performance is improved by ensuring the database has adequate coverage of the states encountered in practice.

4.6 CONCLUSION

In this paper a data-driven technique is proposed to help solve nonlinear optimal control problems. NNOC addresses the major difficulty faced in indirect optimal control — providing tentative values for the unknowns — by retrieving the solutions to problems that have already been solved using brute force methods. The effects of several crucial parameters such as the database size, number of neighbors, and whether to use sensitivity analysis are investigated. Compared with brute-force random restart technique, this method can obtain the global optimal solution an order of magnitude faster and has the potential for real-time application

in nonlinear MPC.

In future work we intend to enhance the suitability of NNOC for real time control of physical systems. Although current results are promising, robustness could be improved in a number of ways. One approach so might use NNOC to calculate a reference trajectory for a trajectory-tracking controller. Or, we could explicitly optimize robust trajectories for use in the database. Future work should also address scalability to higher-dimensional systems as well as state and parameter uncertainty.

CHAPTER 5: LEARNING TRAJECTORIES FOR REAL-TIME OPTIMAL CONTROL OF QUADROTORS

Although NNOC has shown its advantages over random restart in indirect methods, the problems presented in Chapter 4 are toy problems with low-dimensional parameter space. k-NN method does not scale well with problem dimensionality and indirect methods have limited scope of application. In this chapter the neural network is applied on a more challenging problem point-to-point movement of quadcopters solved by direct methods. It turns out the neural network is capable of approximating the argmin function with high accuracy. Different from NNOC where the model prediction is used as initial guess for nonlinear optimizer, here the predicted trajectory is directly used for tracking, after an additional step of local refinement to reduce the error of dynamics constraints. This method generates approximately optimal trajectories without solving nonlinear optimization in real time. The generated trajectory is easier to track compared with trajectories generated with geometry such as minimum-snap trajectory. This claim is proved using physical experiments on a weakly-actuated quadcopter. The real time object tracking experiments show its capability in reactive tasks. [1]

[1] This chapter is reproduced from Gao Tang, Weidong Sun, and Kris Hauser, "Learning trajectories for real-time optimal control of quadrotors". In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

5.1 INTRODUCTION

Trajectory optimization is generally challenging to solve reliably and efficiently. In practice, to explore the benefits of trajectory optimization for agile maneuvers, the trajectories are usually computed offline [127] and many approaches [14, 16] are proposed to improve computational efficiency. As shown in Chapter 1, trajectory learning is a promising approach to get the high-quality trajectory from optimization while avoiding its computational challenges. We propose a technique that can exploit the versatility of trajectory optimization on handling different dynamics, constraints, and cost functions by using machine learning to help avoid its high computational requirement. We adopt the supervised learning approach and explore the ability of neural networks to perform optimal trajectory prediction. To handle the prediction errors made by the neural network, it is important to introduce some postprocessing in order to respect dynamics and control constraints. We show that only a small amount of optimization suffices to achieve high performance. Moreover, for this problem, the discontinuity of argmin does not exist so the model selection and training is quite straightforward.

We evaluate effectiveness of this approach on a quadrotor point-to-point navigation problem. Trained on 50,000 examples, our method calculates near-optimal trajectories in less than 2 ms on average. Experiments demonstrate that trajectories calculated by our technique achieve lower tracking error than minimum-snap trajectories of the same duration [50]. Its fast rate of computation also allows it to be used in a model predictive control (MPC) framework in which the trajectory has to be replanned frequently. Our approach enables agile response to replanning command and the quadrotor is able to stay above a quickly moving target object.

5.2 RELATED WORK

Generation of optimal trajectories for dynamic systems in real-time often requires either simplification of system dynamics into linear systems such as double integrator or parameterization of state trajectories in a limited function space such as piecewise polynomials. This paper is concerned primarily with quadrotor trajectory generation, which has been explored by many other researchers. Pioneered by Mellinger et al. [50], a quadrotor can explore the differential flatness in its dynamics. The trajectory is parameterized using piecewise polynomials and minimizes a combination of the derivatives of the position states and yaw angle, so-called minimum-snap trajectory. These approaches limit the trajectory class to polynomial functions and the available choices of cost functions and constraints are lim-

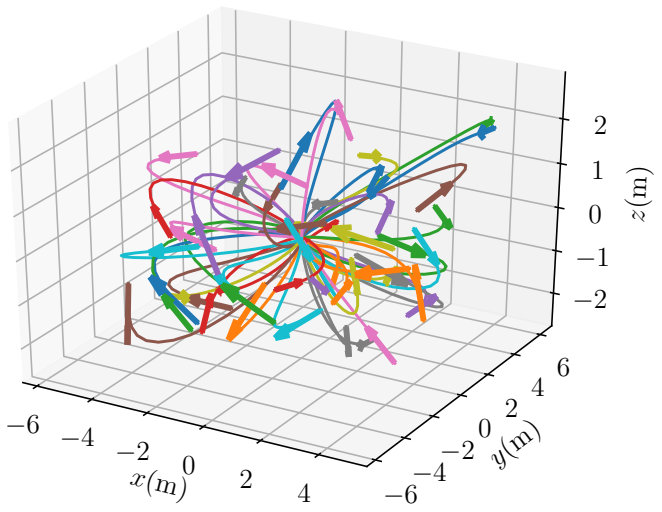


Figure 5.1: Samples of a few optimal trajectories in the quadrotor dataset. The arrow shows initial velocity direction. The target state is fixed at the origin.

ited. Another drawback is these approaches explore the differential flatness of the quadrotor system and cannot be easily extended to quadrotors augmented with slung loads or arms, or more precise model such as considering air drag.

On the other hand, numerical optimal control is versatile and does not require specific system dynamics, cost function, or constraints. In [128] numerical optimal control is demonstrated on a wide range of quadrotor related trajectory optimization problem. However, it needs to solve non-linear programming (NLP) problems [80] which in general cannot be solved in real time or to a global optimum due to high computational expense.

5.3 METHODS

Our approach is composed of four major components.

1. Formulate the problem of interest into a parametric OCP.
2. Generate a training database by sampling parameters from a given range and solving for their optimal trajectories.
3. Use a neural network (NN) to learn the mapping from parameters to optimal trajectories.
4. Online, given a new set of problem parameters, use the NN to predict an optimal trajectory, and then solve a one-step QP to refine the prediction.

Although components 2–3 are computationally expensive, they are only performed once offline. Only component 4 is performed repeatedly online, and we demonstrate that it can be performed extremely quickly.

5.3.1 Parametric Optimal Control

We address dynamical systems in the form

$$\dot{x} = f(t, x, u) \tag{5.3.1}$$

where t is time; $x \in \mathbb{R}^n$ is the state variable; $u \in \mathbb{R}^m$ is the control variable. We refer [50] for the detailed dynamical equations. The state $x = (x, y, z, v_x, v_y, v_z, \phi, \theta, \psi, p, q, r) \in \mathbb{R}^{12}$ and control $u \in \mathbb{R}^4$. The control is directly chosen as the PWM (scaled to $[0, 1]$) for each rotor due to the nonlinear relation between PWM and its thrust and moment [129].

A trajectory is a mapping from time to state and control variables, i.e. $f : t \to [x(t), u(t)]$ with $t \in [0, t_f]$. We use direct transcription approach to solve OCPs. An equidistant time grid of size $N + 1$, i.e. $\{t_i\}_{i=0}^N$ is used for discretization. The trajectory is thus $z = \{t_i, x_i, u_i\}_{i=0}^N$. The cost function to be minimized is

$$J = wt_N + h \sum_{i=0}^{N-1} [{x_i}^{\mathrm{T}} Q x_i + \left(\frac{u_i - u_{i-1}}{h}\right)^{\mathrm{T}} R \left(\frac{u_i - u_{i-1}}{h}\right)] \tag{5.3.2}$$

where h is the grid size; u_{-1} is the nominal control that compensates gravity; w, Q, R penalizes flight time, state, and change of control variables; the term $(u_i - u_{i-1})/h$ approximates $\dot{u}$. We penalize the change of control since our drone has difficulty ramping up its PWM. The penalty on transfer time w encodes the aggressiveness of the trajectory. This cost function is difficult to be directly optimized using the minimum-snap or related approach. We limit $u \in [0.2, 0.85]$ to avoid saturation when feedback is introduced. Again, control bound on PWM is also difficult to be constrained in the minimum-snap framework. Throughout the paper we use $w \in [0.1, 5]$, $Q = \mathrm{diag}(0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1)$, $R = \mathrm{diag}(5, 5, 5, 5)$; System dynamics impose constraints

$$x_{k+1} = \mathrm{RK4}(x_k, u_k, h) \tag{5.3.3}$$

where RK4 means integrating f with constant control u_k for time period h from state x_k. We note that RK4 is used for higher integration accuracy and thus N can be reduced. The

initial and final states specification imposes constraint

$$x_0 = s_0; x_N = s_f \tag{5.3.4}$$

where s_0 and s_f are the desired initial and final states. Additionally, depending on specific problem, other path constraint such as collision avoidance and bounds on state and control can be applied.

The problem is to solve the optimal trajectory from a given initial position and velocity (assuming zero angle and angular velocity) to the origin with zero velocity with different choice of aggressiveness encoded by w. We denote the vector collecting 7 problem parameters (3 initial position, 3 initial velocity, and w) as p. Due to the limitation of the laboratory size, the initial position is limited within $[-5, -5, -2.5]$ and $[5, 5, 2.5]$ and velocity is limited within $[-2, -2, -1.5]$ and $[2, 2, 1.5]$. Since the initial velocity can be non-zero, this parametric OCP provides the flexibility of commanding the quadrotor to another target in flight without stopping. We only consider the obstacle-free problem, and intend to address obstacles in future work.

5.3.2 Learning Optimal Trajectories

The solution to the parametric OCP is a mapping from problem parameter p to the corresponding optimal trajectory $z(p)$. This mapping can be approximated using neural networks. The problem parameter is directly treated as a 7-D vector including 3-D position, 3-D velocity, and time penalty weight w. While the position and velocity parameters enables prediction of optimal trajectories based on current state, w controls the aggressiveness of the predicted trajectory. We assume the angle and angular velocity are small and can be controlled much faster than position and velocity so they are not included in problem parameters to simplify the problem. We encode the solution using a long vector, denoted as Z composed of states $\{x_i\}_{i=0}^{N}$, controls $\{u_i\}_{i=0}^{N-1}$, and time t_N. The neural network is simply chosen as a multilayer perceptron (MLP) with one hidden layer. It takes problem parameter as input and the output is the encoded optimal trajectory, i.e. $g(w, p) : p \to Z(p)$ where w are the weights of the network. Through learning, we find optimal w to minimize

$$L = \mathbb{E}_{p \sim P_{\text{data}}} \text{loss}(g(w, p), Z(p)) \tag{5.3.5}$$

where loss is any regression loss function. We use smoothed L1 loss in this paper.

We train on a dataset of parameter-solution pairs $\{(p^{(1)}, Z(p^{(1)}), \ldots, (p^{(M)}, Z(p^{(M)})\}$ by

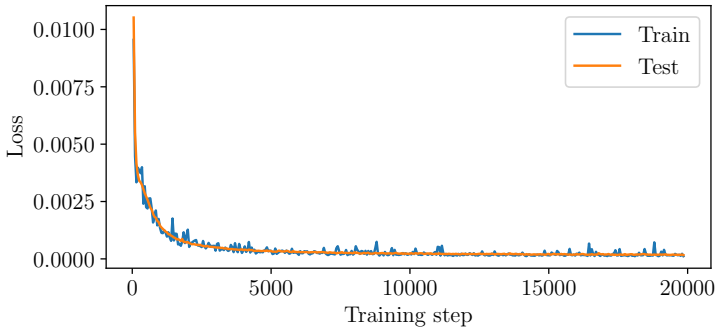


Figure 5.2: History of neural net training and test error

sampling a set of problem parameters $p^{(1)}, \ldots, p^{(M)}$ and solving their corresponding OCPs using a nonlinear programming (NLP) formulation. To generate the training dataset, both position and velocity are sampled uniformly within range, while w is sampled uniformly after log transformation. We firstly solve 5000 problems to optima using random restart with different initial guesses. The random restart technique increases the probability that the solutions to those problems are indeed globally optimal. These 5000 problem-solution pairs are used as database and the NNOC approach [11], which initializes the local optimizer with several nearest neighbors in the database, is used to solve the rest of the problems. The database can be built incrementally. Eventually we collect $M = 50,000$ samples. After the whole dataset is built, we resolve all examples again using NNOC to further reduce the likelihood of local optima in the database. Samples of optimal trajectories are shown in Fig. 5.1.

We train a neural network with input layer of size 7, hidden layer of size 500, and output layer of size 317 (in this paper we choose $N = 20$). The hidden layer has a nonlinear activation function, specifically Leaky ReLU with $\alpha = 0.2$. 80% data are used for model training and the rest is used as test set. Stochastic gradient descent with momentum is used for training with a mini-batch of 64. The training is terminated when the test error does not decrease within 1000 iterations. Fig. 5.2 illustrates the learning curves. At the end of training, test error is 1.7×10^{-4}, which indicates that the network approximates the function accurately.

5.3.3 Refinement by QP

The neural network is capable of predicting $Z(p)$ fairly well for any p, but the prediction might not fully respect all the constraints due to approximation error. Although numerical

simulation shows the violation is low and the prediction can still be used for trajectory tracking, this prediction improved by subsequent optimization. NLPs are often solved using a sequential quadratic programming (SQP) algorithm, but our refinement technique only solves a QP once. We call this approach one-step QP (OSQP), and demonstrate that it is particularly fast when sparsity is exploited. If more computational resources are available, this approach can be extended to perform a small amount of SQP to further refine the trajectories, e.g., performing backtracking line search or trust region approach and multi-step update.

Assuming the prediction Z is decoded into $\{t_i, \bar{x}_i, \bar{u}_i\}_{i=0}^N$, we want to refine this prediction by finding $\delta Z \equiv \{0, \delta x_i, \delta u_i\}_{i=0}^N$ such that $Z + \delta Z$ solves optimal control problem. We note that h is not optimized to keep the problem as QP. As shown later, the prediction error in transfer time t_N is small.

Substituting $Z + \delta Z$ into the cost function and constraints yields

$$\begin{aligned} J = \text{Constant} + h \sum_{i=0}^{N-1} [\delta {x_i}^{\mathrm{T}} Q \delta x_i + 2\bar{x}_i{}^{\mathrm{T}} Q \delta x_i + (\delta u_i R \delta u_i \\ + \delta u_{i-1} R \delta u_{i-1} + 2(\bar{u}_i - \bar{u}_{i-1})^{\mathrm{T}} R(\delta u_i - \delta u_{i-1}))/h^2] \end{aligned} \tag{5.3.6}$$

which is quadratic in terms of δZ and

$$\bar{x}_{k+1} + \delta x_{k+1} = \text{RK4}(\bar{x}_k + \delta \bar{x}_k, \bar{u}_k + \delta \bar{u}_k, h) \tag{5.3.7}$$

which is linearized to

$$\bar{x}_{k+1} + \delta x_{k+1} = \bar{y}_k + \frac{\partial \bar{y}_k}{\partial \bar{x}_k} \delta x_k + \frac{\partial \bar{y}_k}{\partial \bar{u}_k} \delta u_k \tag{5.3.8}$$

where $\bar{y}_k = \text{RK4}(\bar{x}_k, \bar{u}_k, h)$. Since the neural network predicts Z close to the optimum, δZ is small and linearization is valid. We conveniently change the nonlinear dynamics constraints into linear constraints. Additionally, the constraints for initial and final states are readily converted into

$$\bar{x}_0 + \delta x_0 = s_0; \bar{x}_N + \delta x_N = s_f \tag{5.3.9}$$

and other constraints such as bounds on state and control variables can be converted similarly.

We note that the neural network predicts a trajectory with zero angle and angular velocity which might be different from the quadcopter's current state. This issue is further reduced by solving optimal δZ since to satisfy Eq. (5.3.9) the result trajectory has an initial state identical to the quadcopter's current state.

These constraints are assembled into a QP of the form

$$\begin{aligned} \underset{x}{\text{Minimize}} \quad & \frac{1}{2} x^{\mathrm{T}} P x + q^{\mathrm{T}} x \\ \text{subject to} \quad & l \leq Ax \leq u \end{aligned} \tag{5.3.10}$$

with positive semi-definite matrix P. From Eq. (5.3.9) δx_0 and δx_N can be determined directly. The optimization variables for QP are thus $\{\delta x_i\}_{i=1}^{N-1}$ and $\{\delta u_i\}_{i=0}^{N-1}$. Observing that Q and R are both diagonal in Eq. (5.3.6), P is also diagonal. The linear constraints contain linear equality constraints from Eq. (5.3.8) and inequality constraints of bounds on state and control. The bounds on state and control variables are essentially block identity matrix in A. There are N sets of linearized dynamical constraints as Eq. (5.3.8) for $k = 0, \ldots, N-1$. Each instance of (5.3.8) introduces at most $n \times (n+m+1)$ nonzero elements. Out of those nonzero elements, n^2 belong to $\frac{\partial \bar{y}_k}{\partial \bar{x}_k}$, nm belong to $\frac{\partial \bar{y}_k}{\partial \bar{u}_k}$, and the rest n is the diagonal matrix associated with δx_{k+1}. To exploit this sparsity, we use the implementation in [130], and all tested problems can be solved in microseconds.

5.4 RESULTS

In this section, we evaluate the method's performance in simulation and on a real quadcopter.

5.4.1 System Description

We use a commercially available quadrotor Crazyflie 2.0[2], with basic specifications listed in Tab. 5.1. We refer to [129] for more details on system dynamics. The position of the quadrotor is captured by the Vicon motion capture system and transmitted to ground control station using Ethernet at 200 Hz. The raw data stream from Vicon goes through a Kalman filter and then serves as feedback for a position controller on the ground control station. The position controller is a proportional-integral-differential (PID) controller, and sends commands at 100 Hz through radio to the quadrotor which drives the on-board attitude controller at 500 Hz. The commanded target position is fed into the neural network to generate a trajectory and then optimized by OSQP to get the optimal trajectory. The optimal trajectory is then sent to the position controller as a reference trajectory. The system diagram is shown in Fig. 5.3.

[2] https://www.bitcraze.io/crazyflie-2/

Table 5.1: Specifications of Crazyflie2.0

Parameter	Value
Mass (with markers)	33.5 g
Size(W×H×D)	92×29×92 mm
Takeoff weight	42 g
Flight time	7 min

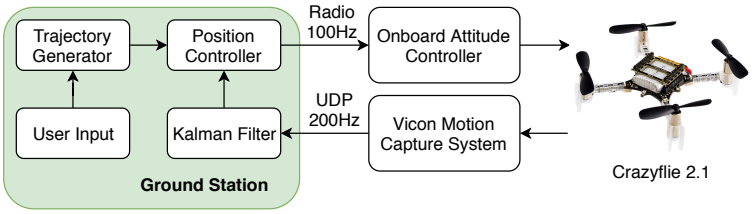


Figure 5.3: Architecture of the system

5.4.2 Numerical Validation

The top row of Fig. 5.4 shows the prediction of optimal trajectories from $(-3, -3, -2)$ with zero velocity to the origin with different weights on transfer time. These show that the chosen aggressiveness affects the optimized transfer time. The bottom row indicates that the neural network makes accurate predictions. Refinement by QP is able to further optimize the trajectory, and the result is visually indistinguishable from the global optimum.

We evaluate the network's prediction error in transfer time in Fig. 5.5 and Tab. 5.2. It shows that most of the errors are within 0.25 s. This is important because OSQP is unable to improve the transfer time. However, there are still a few outliers with large transfer time error. These tend to be non-aggressive problems. For example, in the worst problem, $w = 0.15$ and the costs from the OSQP and optimal trajectory are 0.870 and 0.739, respectively.

Next, we examine the violation of the dynamics constraint. We randomly sample 1000 initial states and evaluate the L2 norm of the violation of constraints of the optimal tra-

Table 5.2: Prediction error in transfer time (s)

MAE	RMSE	max	median
0.056	0.080	1.62	0.043

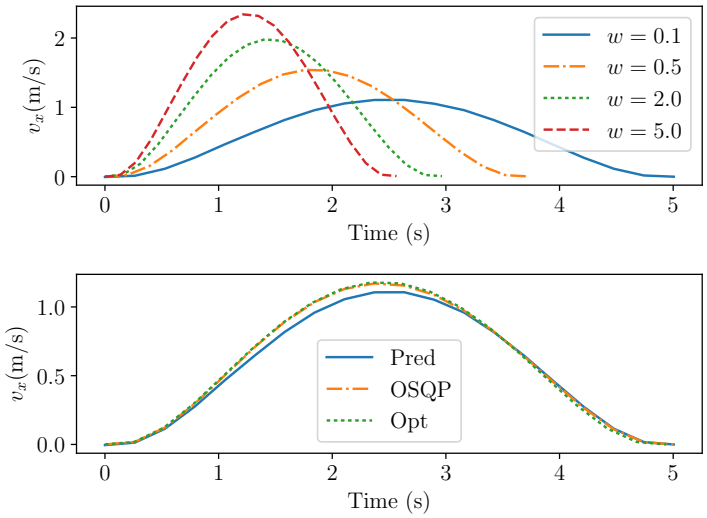


Figure 5.4: Top: four trajectories from the same initial state with different aggressiveness weights. Bottom: predicted, OSQP refined, and optimal trajectories for $w = 1$. The curves almost overlap, indicating high prediction accuracy.

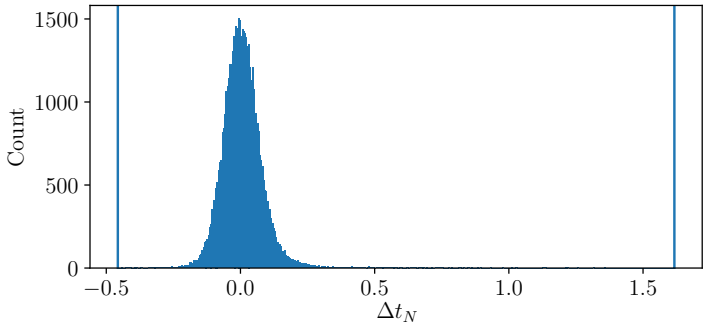


Figure 5.5: Prediction error in transfer time. Most prediction errors are within 0.25 s. Outliers (invisible in histogram) are indicated with vertical lines.

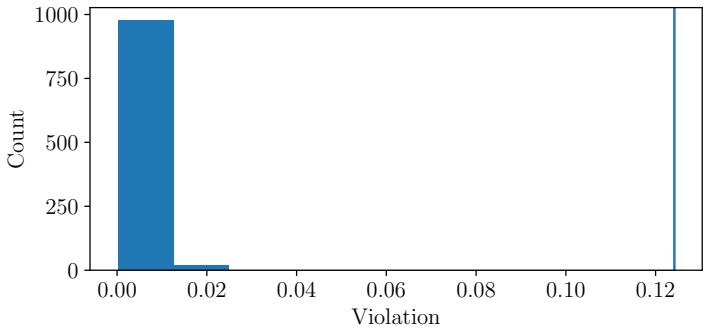


Figure 5.6: Histogram of constraint violation measured by the norm of the constraint function. Outliers (invisible in histogram) are indicated with vertical lines.

jectories from our approach (0 means no violation). The result is shown in Fig. 5.6 with a worst case of 0.12. Considering that this problem has $(N-1)n = 228$ constraints, even the worst-case violation is relatively small and can be well compensated by feedback control.

Tab. 5.3 compares the running time and cost of the following methods:

1. Our approach (NN+OSQP)
2. Minimum-snap trajectory (Min-Snap) [50]
3. NLP solver with straight line initialization (SL+NLP)
4. NLP solver with NNOC initialization (NNOC) [11]
5. NLP solver with neural net initialization (NN+NLP)

on 1000 sampled initial states. We note that Min-Snap is implemented in Python and a careful C++ implementation can reduce the computation time to the same level with our approach. The transfer time of minimum-snap trajectory must be set by some other methods which often leads to conservativeness for safety. In these experiments it is selected to be the same with the results from NNOC. Since the minimum-snap approach is optimizing the snap of selected state variable, it will have larger cost for our cost function.

Compared with the full NLP solver, our approach is able to get an approximate solution *two orders of magnitude* faster at similar levels of cost. It also obtains better cost than Min-Snap. Although we are not explicitly optimizing control energy, it turns out our approach yields lower energy trajectory. Besides, Min-Snap has to check violation of control constraint a posterior and as a result, the transfer time has to be chosen conservatively in practice.

[3]See Eq. (5.3.2)

Table 5.3: Comparison between approaches.

	NN+OSQP	Min-Snap	SL+NLP	NNOC	NN+NLP
Success	1000	1000	563	1000	1000
Time (ms)	1.80	10.21	382.9	194.7	131.1
Avg. cost [3]	8.73	8.88	8.64	8.64	8.64
Avg. energy	6.67	7.00	6.69	6.69	6.69
Avg. jerk	0.37	0.40	0.35	0.35	0.35

5.4.3 Point-to-point Navigation and Real-time Tracking

Fig. 5.7 compares our method applied to the real quadcopter by a point-to-point maneuver from (0, 0, 0) to (3, 3, 1.5). Our approach predicts a transfer time of 3.1 s. The minimum-snap trajectory to reach the same target within the same amount of time is also shown. The two reference trajectories are quite different, especially in the z direction. This is not too surprising because they are optimizing different cost functions. The trajectory from our prediction yields better tracking performance. For underpowered drone like Crazyflie, the specialized cost function in Eq. (5.3.2) penalizes rapid change of rotor PWM so it leads to better performance than the general minimum-snap approach.

We repeated this experiment for 10 manually selected targets. The tracking errors, measure by the norm of position and velocity errors, are listed in Tab. 5.4. It shows our approach generates a trajectory that is easier to track than the minimum-snap approach, given the same transfer time.

Table 5.4: Comparison of average tracking error

NN+OSQP	Min-Snap
0.45	0.87

Since our approach can predict a trajectory with non-zero initial velocity, it can switch target rapidly. Fig. 5.8 demonstrates this capability. A movement to the initial target is interrupted with a new target after 1.7 s. The technique smoothly and immediately switches to the new trajectory.

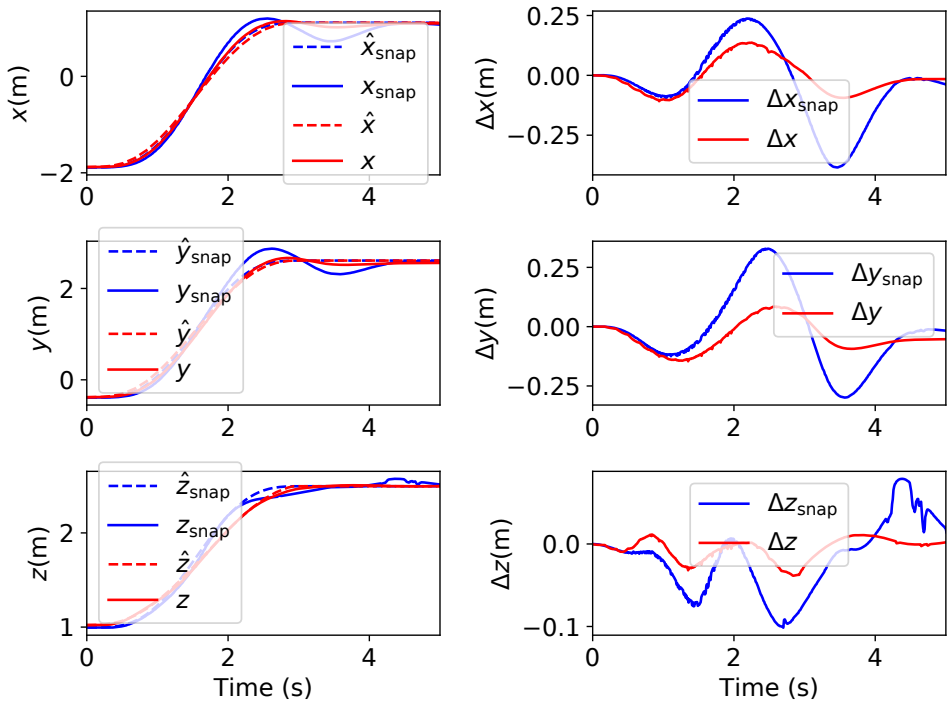


Figure 5.7: Results for tracking a trajectory. The dashed and solid lines are reference and actual trajectory. The blue and red lines are the trajectory by our minimum-snap and our approach. To reach the same target within the same amount of time, our approach generates a trajectory with better tracking performance.

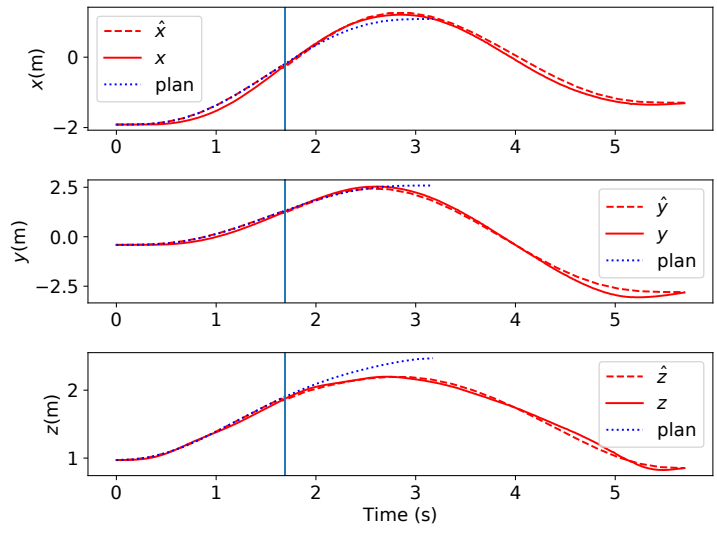


Figure 5.8: Results for replanning during tracking. The vertical line indicates when replanning is commanded. The blue curve shows the planned trajectory to the first target. Our approach is able to generate optimal trajectory in real time.

5.5 CONCLUSION

We exploit the ability of machine learning for global nonlinear function approximation and efficiency of local trajectory optimizer to enable real-time OCP solving.

The problem of interest is formulated as parametric OCP so dataset of optimal trajectories is created and the parameter-solution mapping can be learned. It turns out the optimal trajectories can be learned using small amount of data and approximated to high precision. The local trajectory optimizer based on sparse QP benefits from the high accuracy of the prediction from the learned model. The combination of these techniques enables real-time solving of challenging nonlinear OCPs. We validate this approach using an indoor quadrotor system. We note that this quadrotor is under-powered and light-weight so its disturbance rejection capabilities is relatively weak. In future work we intend to apply our technique to quadrotors capable of more aggressive maneuvers. Our method should benefit more from the exploitation of nonlinear dynamics to achieve higher performance. Future work includes applying this technique to more challenging systems such as locomotion and theoretical study of stability guarantee and bounds on loss of cost function.

CHAPTER 6: DISCONTINUITY-SENSITIVE OPTIMAL CONTROL LEARNING BY MIXTURE OF EXPERTS

In Chapter 5 standard neural network (SNN) emerges as a model for trajectory learning with better scalability to parameter dimensionality and generalization to unseen problems. However, SNN is a continuous model and has difficulty handling discontinuity which exists in the argmin function of lots of parametric planning tasks. This chapter studies how to learn discontinuous argmin function using the Mixture of Experts (MoE) model which contains a classifier that makes discontinuous decisions. The training method for MoE is proposed which outperforms standard training method for SNN using backpropagation. To train MoE, the dataset has to be split into groups within which the argmin is continuous. The k-Means clustering approach with carefully tuned k splits the dataset reasonably well with the help of several metrics to measure the discontinuity within each split. Numerical experiments on benchmark problems show the advantages of MoE over SNN in terms of success rate for trajectory tracking. [1]

[1] This chapter is reproduced from Gao Tang and Kris Hauser, "Discontinuity-Sensitive Optimal Control Learning by Mixture of Experts". In 2019 IEEE International Conference on Robotics and Automation (ICRA).

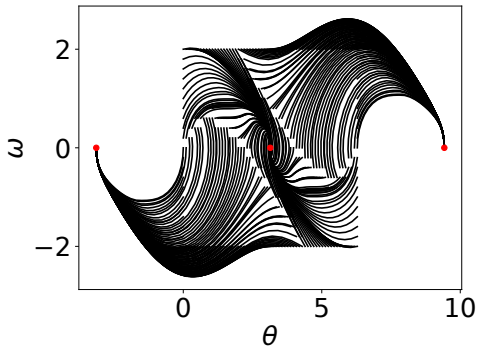
6.1 INTRODUCTION

An underappreciated challenge for learning argmin from data is that function approximators, such as Standard multilayer feedforward Neural Networks (SNN), perform poorly near discontinuities that are prevalent in many nonlinear OCPs. Fig. 6.1 shows the results of learning a pendulum swingup task by SNN from optimal trajectories. There exist three possible goals among all the optimal trajectories so the problem-optimum mapping is not globally continuous. Although neural networks are capable of approximating continuous nonlinear functions [131], near the region where the optimal goal state switches, they tend to predict an "average" of the two goals, as shown in Fig. 6.1b. Even with a large amount of data sampled densely near the boundaries, a given network may not have the capacity to represent such a function accurately, and even for high-capacity deep networks it may be difficult for training to converge.

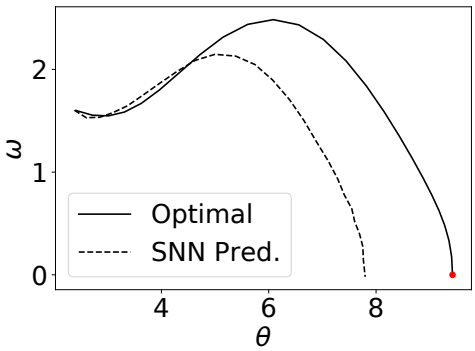
This chapter addresses this problem by modifying the Mixture of Experts (MoE) [132, 133, 134] model to learn discontinuous argmin function. Given a problem parameter, MoE first uses a classifier (gating network) to select a regressor (expert) and then use the regressor to make a prediction (Fig. 6.2). MoE is trained such that each regressor works in a region of the parameter space where the problem-optimum mapping is continuous. This is reminiscent of a divide and conquer approach, which has already been widely used in the control community for controller design [135]. Fig. 6.1c illustrates that the pendulum swingup dataset can be divided into three regions, and with this division MoE makes better predictions than SNN particularly near discontinuities, as shown in Fig. 6.1d.

We propose a training approach for MoE that significantly outperforms backpropagation [134] or expectation maximization [133] on the whole model. The training data is a collection of (problem parameter, optimal trajectory) pairs sampled and optimized in a preprocessing step. We first partition the data such that within each cluster, the problem-optimum mapping is continuous. Then the classifier is trained to predict the identity of the partition and a separate regressor is trained for each partition. Each component is trained individually using backpropagation.

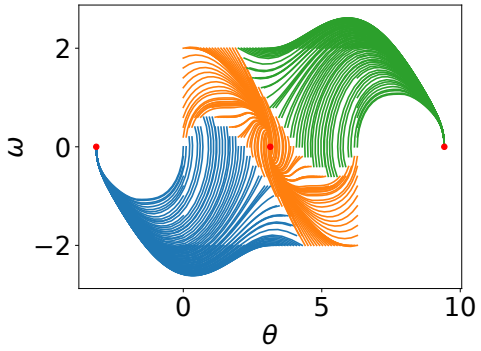
In this chapter, clustering and data split are used interchangably. A proper clustering should eliminate intra-cluster discontinuity. We find that the combination of PCA and k-Means is applicable in a wide range of benchmark problems and overall provides reasonable performance. We also study how performance varies as the number of clusters k increases. We present a tuning method that chooses the clustering hyperparameters to maximize intra-cluster continuity. We propose and compare a number of metrics based on trajectory distance between neighbors, rate of trajectory change given parameter change, and constraint



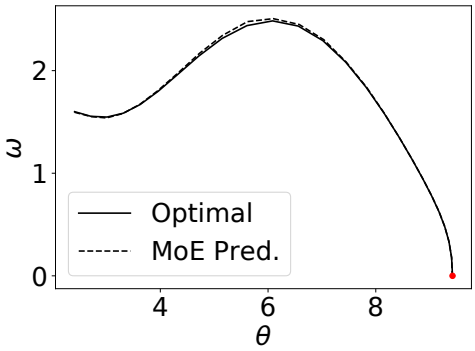
(a) Samples of data



(b) SNN Prediction



(c) Samples of clustered data



(d) MoE Prediction

Figure 6.1: Illustration of dataset and prediction of a selected state from SNN and MoE for the pendulum swingup task (see Sec. 6.4.1 for details). (a) Samples of optimal pendulum swingup trajectories. The red circles are possible target states. (b) SNN (trained using data in (a)) prediction of a selected state. The solid and dashed lines denote the optimal and predicted trajectories, respectively. (c) Samples of clustered optimal trajectories where each color denotes one cluster. Trajectories are manually clustered according to the final states. (d) MoE prediction to the same state as (b). Compared with SNN, MoE does not predict a trajectory which averages trajectories from two clusters.

violation of the average trajectory of neighbors. These metrics provide a "fingerprint" of clustering performance which aids an engineer in comparing among possible clustering assignments. We also test a few off-the-shelf clustering metrics and show they do not provide insights on data discontinuity.

For online use, the MoE-predicted trajectory is combined with a trajectory tracking controller to accomplish the given task. Experiments on selected benchmark problems demonstrate that suitably trained MoE models can learn near-optimal trajectories suitable for trajectory tracking with remarkably high success rates (99.5+%).

The contributions of this chapter are:

1. The MoE model is proposed to learn the discontinuous problem-optimum mapping and the training methods are compared.
2. A simple yet effective method is proposed to cluster the trajectories.
3. Clustering metrics are proposed to tune clustering hyperparameters and they agree with empirical results.
4. Test of the learning pipeline on a sensor-based navigation problem with high input dimension.

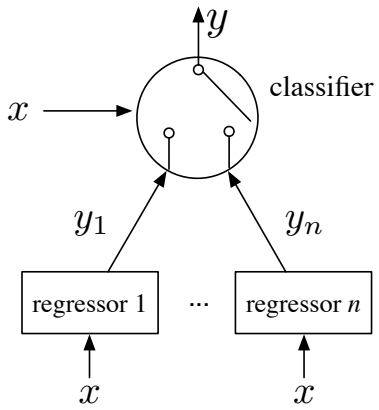


Figure 6.2: Illustration of MoE. For every problem the prediction is made by *one* out of n regressors selected by the classifier.

6.2 RELATED WORK

The discontinuity of the problem-optimum mapping has long been known [65], a fact that has been underappreciated in the control learning community. Under certain assumptions,

this function is piecewise continuous, and discontinuity-tolerant methods have been proposed for learning from optimal solutions [11, 114]. However, their approaches do not explicitly try to partition the space into regions and the k-NN model does not scale well to input dimensionality. In contrast, the MoE model does indeed requires the segmentation the dataset since it is a parametric model and has better generalization capability.

Approximation of piecewise continuous function using neural networks has been studied in [136, 137]. In [136] discontinuous activation function is used but the finite number of discontinuity points have to be known. In [137] a constructive method is used to build the neural network. It requires precise knowledge of where discontinuity occurs and is thus not practical in many applications. Learning discontinuous function with high dimensional input and output purely from data without prior knowledge of discontinuity has not received lots of research attention.

Most closely related work is previous research on MoE [132, 133, 134, 138]. Recent work in [139] proposed a novel cost function to learn MoE on human-driven vehicle trajectories, but this approach is sensitive to model initialization based on our experiments. This chapter proposes several modifications to make MoE suitable for learning optimal trajectories. We use hard classification boundaries to avoid predictions that tend to be an average of the boundary's two sides. Another significant difference is the modification to the training approach. Traditionally MoE is trained using either backpropagation [134] or expectation maximization [133] so the gating function (i.e. classifier) and experts are simultaneously updated. However, we train the classifier and regressors independently, and experiments suggest that this significantly increases trajectory tracking success rate.

6.3 METHODOLOGY

In this section, the problem of learning from optimal control is formulated and the key components are analyzed. The proposed approach addresses a parametric OCP using the following procedure:

1. Input: sample OCP parameters and collect dataset of problem-optimum pairs.
2. Cluster: select a clustering approach to cluster the trajectories, which also partitions the parameter space.
3. Train: train the classifier and regressors individually using backpropagation.
4. Validate: predict optimal trajectories for the validation set and perform trajectory rollout.

6.3.1 Parametric Optimal Control

In many applications, trajectory optimization problem is parameterized (e.g. by initial and final states) and many realizations of parametric problems have to be solved, especially for reactive tasks. Parametric OCP is formulated similar to Eq. (1.1) but every component including dynamics, cost function, and constraints may be parametrized by some parameter p. The solution is also a function of p. Parametric OCP is generally difficult to solve analytically [123], but for any given parameter, numerical methods may be used to solve the resulting OCP as what is done in previous chapters.

Supposing the problem is parameterized by p, the parameterized trajctory optimization problem is

$$\begin{aligned} \underset{x,u,T}{\text{minimize}} \quad & J(p,x,u,T) = \int_0^T \ell(p,x,u,t)dt + \Phi(p,x(T)) \\ \text{subject to} \quad & \dot{x} = f(p,x,u) \\ & x(0) \in \mathcal{X}_0(p), x(T) \in \mathcal{X}_f(p) \\ & x(t) \in \mathcal{X}(p), u(t) \in \mathcal{U}(p) \quad \forall t \in [0,T] \\ & \phi(p,x,u) \leq 0, \quad \forall t \in [0,T] \end{aligned} \tag{6.3.1}$$

where parameter p may appear in places ranging from objective function, system dynamics, control feasible set, to path constraint, showing the flexibility of parameterized optimization. For each realization of the parameterized problem, methods to solve problems defined in Eq. (1.1) are still valid to use. Using direct transcription as an example, the time interval $[0,T]$ is evenly divided using a grid of size $N+1$, and the state and control are discretized using the grids, as $\{t_i, x_i, u_i\}_{i=0}^N$ where $t_i = iT/N, x_i = x(t_i), u_i = u(t_i)$. The problem in Eq. (6.3.1) is discretized into

$$\begin{aligned} \underset{x_i,u_i,T}{\text{minimize}} \quad & J(p,x,u,T) = h\sum_{i=0}^{N-1} \ell(p,x_i,u_i,t_i) + \Phi(p,x_N) \\ \text{subject to} \quad & (x_{i+1}-x_i)/h = f(p,x_i,u_i), \quad i = 0,\cdots,N-1 \\ & x_0 \in \mathcal{X}_0(p), x_N \in \mathcal{X}_f(p) \\ & x_i \in \mathcal{X}(p), u_i \in \mathcal{U}(p) \quad i = 0,\cdots,N \\ & \phi(p,x_i,u_i) \leq 0, \quad i = 0,\cdots,N \end{aligned} \tag{6.3.2}$$

where $h = T/N$; $\dot{x}$ is approximated using forward finite difference and for direct collocation such approximation is different [81]. In this formulation, the unknowns are discretized states and controls on the grid (and total time T for problems with free duration), denoted as $z \equiv [\{x_i\}_{i=0}^N, \{u\}_{i=0}^N, T]$ and the constraints are discretized version of constraints in Eq. (1.1). The optimum of Eq. (6.3.2) is denoted as $z^*(p)$. Essentially a numerical optimization problem with finite number of unknowns and constraints described above is solved to obtain the

optimal trajectory.

In this chapter the nonlinear optimization problem and solves it using SNOPT [37]. Our goal is to approximate the argmin function $p \to z^\star(p)$. The goal is to learn a function $z : \mathbb{R}^l \to \mathbb{R}^R$ that approximates $z(p)$ where R is the length of vector z.

6.3.2 Optimal Trajectory Database Generation

To train models we generate a database of optimal trajectories $z_1, \ldots, z_M$ to sampled problems $p_1, \ldots, p_M \in \mathbb{R}^l$ where M is the data size. Due to non-convexity, even finding a global optimum to a single problem can be difficult. One practical approach is to pick the best local optimum from a multi-start method. However, the local optimum can be also quite difficult to find if an initial guess not close to the optimum is provided. We adopt a nearest-neighbor approach in Chapter 4 to help generate large databases quickly. We first sample some number of problems (fewer than M but much larger than the number of expected partitions) and use an exhaustive random restart approach to solve them. These solutions are used as the initial database. Then we sample more parameters, and for each new problem we attempt local optimization from each of its k-nearest neighbors to find k local optima. The best solution is kept in the database. We note that this process is done completely offline and parallelizable.

6.3.3 Mixture of Experts

The MoE model is composed of a classifier and r regressors, as shown in Fig. 6.2. In this paper both models are chosen as multilayer feedforward neural networks. Each regressor takes input $p \in \mathbb{R}^l$ and makes a prediction $y_i(p, w_i) \in \mathbb{R}^R, i = 1, \ldots, r$ where w_i specifies the regressor weights. The classifier, with weights w_c, takes input p and predicts r values $\{c_i\}_{i=1}^r$. The output of the classifier are combined with softmax to assign probabilities for each model, i.e.

$$P_i = \frac{\exp c_i}{\Sigma_{i=1}^r \exp c_i} \tag{6.3.3}$$

or argmax to select one model only (in this case, $P_k = 1$ for $k = \arg\max_i c_i$ and $P_k = 0$ otherwise.) The difference between softmax and argmax is softmax tends to give a prediction that is a mixture of predictions from all experts. Argmax, however, selects one model and ignores other models' predictions. The choice of softmax and argmax is compared on a benchmark problem. It turns out argmax is a better choice for trajectory learning and this is a fundamental difference from [132] where softmax is used.

In either case, the final prediction is a mixture of predictions from all regressors, i.e.

$$z(p) = \Sigma_{i=1}^{r} P_i(p, w_c) y_i(p, w_i) \tag{6.3.4}$$

The target is to find w_c and $\{w_i\}_{i=1}^{r}$ in order to minimize

$$L = \mathbb{E}_{p \sim P_{\text{data}}} \text{loss}(z(p), z^{\star}(p)) \tag{6.3.5}$$

where P_{data} is a distribution over problems and $\text{loss}(\cdot, \cdot)$ is any regression loss function.

6.3.4 Joint Training

The most straightforward way to train MoE is to treat the joint network as in Eq. (6.3.4) as an SNN, randomly initialize weights, and minimize (6.3.5) using backpropagation. Although several heuristics have been proposed to train MoE using backpropagation such as [134], training may still be unstable. If softmax is used, all the data is used to train each regressor, with weights equal to the probabilities predicted by the classifier. In the case of argmax, each regressor is only trained using data assigned to it by the classifier. There is no gradient to update the classifier if argmax is used. Softmax, on the other hand, can still have gradient to update the classifier.

Since argmax is the limit of softmax if we scale $\{c_i\}_{i=1}^{r}$ by a large positive scalar, we introduce $\epsilon \in [0, \infty)$ which is used to divide the output of the classifier before applying softmax, i.e.

$$P_i = \frac{\exp(c_i/\epsilon)}{\Sigma_{i=1}^{n} \exp(c_i/\epsilon)}. \tag{6.3.6}$$

As $\epsilon \to 0$, the softmax weights approach the argmax function. Hence, ϵ must be gradually lowered to balance between updating weights of classifier and restricting mixture of outputs from multiple regressors. In this paper, we call training all components of MoE directly using backpropagation as joint training while decoupled/individual/independent training means training each regressor and classifier independently.

6.3.5 Trajectory Clustering

As we shall show later, joint training of MoE may improve the loss function compared to decoupled training, but appears to be detrimental to trajectory tracking performance. Clustering has been shown to be effective to avoid some instability in MoE training [138] by training the classifier and regressors of MoE individually on subsets of the data. We adopt

the same approach here, and study how to cluster trajectories such that in each cluster the problem-optimum mapping is continuous.

The dataset $\{(p_j, z_j)\}_{j=1}^{M}$ is divided into r groups $C_1, \ldots, C_r$, ideally so that $z^\star(p)$ is a continuous function for all p in a given region. This problem can be formulated as a clustering problem and each cluster denotes a region of the partitioned parameter space. The classifier is trained to predict $P_i(p_j, w_c) = 1$ for all p_j in C_i, and the i'th regressor is trained as usual, restricted to the examples in C_i. We call this process (decoupled) *pretraining*. In fact our approach terminates after pretraining, but we also study what happens if additional training steps are taken.

We note that since the parameter and its solution are paired in each example, clustering either one is sufficient to cluster the dataset. The trajectories for two problem parameters can be very different even if the problems are close, therefore the trajectories themselves provide better information for clustering. Hence, we experiment with using the distance between optimal trajectories to classify the family of solutions. The simplest approach is to apply standard clustering techniques, such as the k-Means algorithm to trajectories. However, the k-Means algorithm relies on Euclidean distance but the trajectory vector usually has a high dimensionality of up to several hundred. As a result, dimensionality reduction such as PCA is performed before k-Means. We note that mini-batch k-Means is scalable to data size [140] and suitable for our task. Our experiments show the combination of PCA and k-Means results in reasonable results in all the benchmark problems. In order to determine the best k, we propose clustering metrics that reveals function discontinuity within a cluster. The metrics are compared with existing clustering and segmentation metrics. Trajectory rollout performance on the test set is compared with clustering metrics as well.

6.3.6 Function discontinuity

It is beneficial to mathematically define continuity and connectedness.

Definition 6.1. Continuity: For a mapping $f : p \to z$ where $p \in \mathbb{P} \subseteq \mathbb{R}^l$ and $\mathbb{P}$ is open; $z \in \mathbb{R}^R$ and two distance functions d_p and d_z defined on $\mathbb{R}^l$ and $\mathbb{R}^R$, f is continuous if $\forall p \in \mathbb{P}, \forall \epsilon > 0, \exists \delta > 0$ such that $\forall p' \in B_\delta(p) \cap \mathbb{P}, d_z(p', p) < \epsilon$ where $B_\delta(p) = \{p' | d_p(p, p') < \delta\}$. Similarly, f is discontinuous if $\exists p \in \mathbb{P}, \epsilon > 0$ such that $\forall \delta > 0, \exists p' \in B_\delta(p)$ such that $d_z(p, p\prime) > \epsilon$.

Definition 6.2. Path connectedness: We define a path between two points x_i and x_j of a topological space X as a continuous function $x(s)$ from unit interval $[0, 1]$ to X with

$x(0) = x_i, x(1) = x_j$. Space X is path-connected if there is a path joining any two points in X.

Definition 6.3. Set connectedness: Two sets S_1 and S_2 are disjoint if x_i and x_j are not path connected $\forall x_i \in S_1, \forall x_j \in S_2$. A set S is disconnected if it is the union of two disjoint nonempty sets. A connected component is a maximal connected subsets. The union of every connected component is the set S.

The number of connected components is an important set property in this chapter. In some problems, p is from a set with only one connected component while z forms a set of several connected components. The existence of local optima families causes the existence of multiple connected components for z. The mapping f from one connected component to several connected components is surely discontinuous. A typical example is the pendulum swingup problem as shown in Fig. 6.1 where the optimal trajectories form 3 connected components.

However, even if z is also in a single connected component, f might still be discontinuous. To explain this phenomenon, we use an obstacle avoidance problem, resembling the drone with obstacle problem. Considering the shortest problem between two points s and t on x-axis while avoiding a disk (no thickness) obstacle with radius 1 located at $[0.5, u, v]$ and perpendicular to x-axis, as shown in Fig. 6.3. The shortest path will pass the boundary of the disk and that point is sufficient to represent the path (the red point in Fig. 6.3). Solving the geometric problem gives the optimal point $[0.5, u(1-1/\sqrt{u^2+v^2}), v(1-1/\sqrt{u^2+v^2})]$ whose 2nd and 3rd component are both continuous function of u and v except the singularity point at $u = 0, v = 0$. The singularity arises from the non-uniqueness of solution. The solution set has a single connected component. However, the mapping is not continuous due to the existence of singularity.

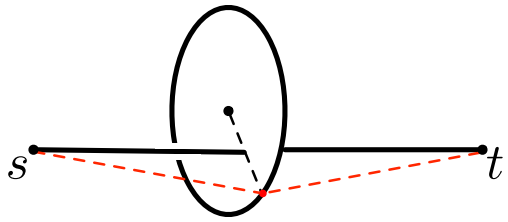


Figure 6.3: Shortest path problem with disk obstacle in 3D.

In this paper, the former two types of function discontinuity both exist. Their different behaviors are described later. A topological view of this type of discontinuity is in Chapter 7.

6.3.7 Trajectory Rollout and Model Validation

While the trajectory learning problem is essentially a regression problem, it is more informative to study the success rate of applying the predicted trajectory as a reference for trajectory tracking controller such as PID, LQR, and MPC. In this paper, an LQR controller is designed to track the trajectory. After trajectory tracking is finished, the robot is expected to be close to an equilibrium point which is further controlled by a stabilizing controller. We call the process of trajectory prediction, tracking, and equilibrium stabilizing as *trajectory rollout*. A rollout is successful if the resulting state converges to a desired equilibrium point.

6.4 BENCHMARK PROBLEMS DESCRIPTION

We study several benchmark problems with increasing dimensionality in problem parameter. A summary is listed in Tab. 6.1.

Table 6.1: Summary of benchmark problems

	pendulum	ground vehicle	drone-obstacle(1)	drone-obstacle(2)	navigation
State dims	2	4	12	12	4
Control dims	1	2	4	4	2
Problem param.	$x_0 \in \mathbb{R}^2$	$x_0 \in \mathbb{R}^4$	$\mathbb{R}^7$	$\mathbb{R}^{11}$	$\mathbb{R}^{74}$
Trajectory dims	75	149	317	317	41
Param range	$[-\pi, \pi] \times [-3, 2]$	$[-11, 10]^2 \times [-\pi, \pi] \times [-3.1, 3.1]$	$[-10, 10]^6 \times [1, 5]$	$[-10, 10]^9 \times [1, 5]^2$	$[0, 0.5]^{72} \times [-1, 1]^2$
Dataset size	1,281	120,009	189,990	454,635	99,995
SNN size	(2, 300, 75)	(4, 200, 200, 149)	(7, 500, 500, 317)	(11, 2000, 2000, 317)	(74, 500, 500, 500, 500, 500, 41)
Validation size	1000	10000	4728	9106	1000

6.4.1 Pendulum Swing-Up

The system dynamic equations are

$$\dot{\theta} = \omega, \dot{\omega} = u - \sin\theta \tag{6.4.1}$$

where the state $x = [\theta, \omega]$ are the angle and angular velocity of the pendulum; $u \in [-1, 1]$ is the control torque. The problem parameters are the initial states. The target state is the straight up state, i.e. $\omega_f = 0, \mod(\theta_f, 2\pi) = \pi$. The cost function is a weighted sum of

time and control energy, i.e.

$$J = w(t_f - t_0) + r \int_{t_0}^{t_f} u^2 \mathrm{d}t \tag{6.4.2}$$

with $w = 1, r = 1$.

The parameter space is a subset of $\mathbb{R}^2$ and we directly sample parameters on a uniform grid of 61×21. The validation set is sampled at random. Samples of optimal trajectories are shown in Fig. 6.1.

Trajectory rollout is performed by first tracking a predicted trajectory using LQR and then applying a stabilizer at the equilibrium for 5 s: we define success as the final state deviates from the equilibrium within 0.1.

6.4.2 Ground Vehicle

We use a planar car with dynamic equations

$$\dot{x} = v \sin\theta,\ \dot{y} = v\cos\theta,\ \dot{\theta} = u_\theta v,\ \dot{v} = u_v \tag{6.4.3}$$

where the state $x = [x, y, \theta, v] \in \mathbb{R}^4$ includes the planar coordinates, orientation, and velocity of the vehicle; the control $u = [u_\theta, u_v]$ includes the control variables which change the steering angle and velocity, respectively. The problem parameters are the initial states within some range and the goal is to control the system to the origin with zero velocity and mod $(\theta_f, 2\pi) = 0$. The cost function is a weighted sum of time and control energy, i.e.

$$J = w(t_f - t_0) + \int_{t_0}^{t_f} r_1 u_\theta^2 + r_2 u_v^2 \mathrm{d}t \tag{6.4.4}$$

with $w = 10, r_1 = r_2 = 1$.

The training and validation dataset are both generated by uniformly sampling the parameter space. Similar to the pendulum swingup problem, the constraint on θ_f makes it possible to reach the goal with different θ_f. One might intuitively think the trajectories have 3 clusters subject to different θ_f. However, later results show 3 clusters are not sufficient to characterize the structure of trajectories. Trajectory rollout is performed by tracking the predicted trajectory using LQR. We note that a stabilizer is not used since the linearized system is not controllable at the target state. As a result, rollout success is determined by whether the norm of system state after trajectory tracking is within 0.5.

6.4.3 Drone with One Spherical Obstacle

The system has state $x = (x, y, z, v_x, v_y, v_z, \phi, \theta, \psi, p, q, r) \in \mathbb{R}^{12}$ and control $u \in \mathbb{R}^4$. The states include 3D position, 3D velocity, 3D Euler angles, and 3D angular velocity and the controls are speeds of the four motors. We refer to Ref. [141] for details of system dynamics of the drone system. The goal is to control the drone from any equilibrium state with position within $[-10, 10]^3$ and all other states zero to the goal state 0. The cost function is a weighted sum of time, control energy, and penalty on states, i.e.

$$J = w(t_f - t_0) + \int_{t_0}^{t_f} x^{\mathrm{T}} Q x + u^{\mathrm{T}} R u \mathrm{d}t \tag{6.4.5}$$

with $w = 10$, $Q = \mathrm{diag}(0, 0, 0, 1, 1, 1, 0.1, 0.1, 0.1, 1, 1, 1)$, $R = \mathrm{diag}(1, 1, 1, 1)$.

One spherical obstacle is considered which imposes additional path constraints on state variables. The obstacle is characterized by its position of center and radius. We uniformly sample initial positions of the drone. However, if we uniformly sample the center and radius of obstacles, in most cases the obstacle is too far to have any impact on the trajectories. Therefore we randomly sample obstacles with radius within $[1, 5]$ around the straight line connecting the initial and final positions. We also reject obstacles that are less than 0.5 to the start and goal positions or has a penetration depth smaller than 0.5 with the straight line between the start and goal. Trajectory rollout is performed by LQR tracking followed by LQR stabilizing controller. It turns out the drone can always be controlled to the target position, with some violation of the obstacle avoidance constraint. The rollout metric is thus chosen as the average constraint violation. This violation can be used to choose an obstacle radius inflation value to avoid collision.

We note that the discontinuity of this problem is caused by singularity (similar to Fig. 6.3) rather than isolated connected components.

6.4.4 Drone with Two Spherical Obstacles

This problem is similar to the previous one and the only difference is two obstacles are considered. It increases the problem parameter dimensionality to 11. The structure of optimal trajectories is similar to the drone problem with one obstacle.

6.4.5 Indoor Navigation with Distance Sensor

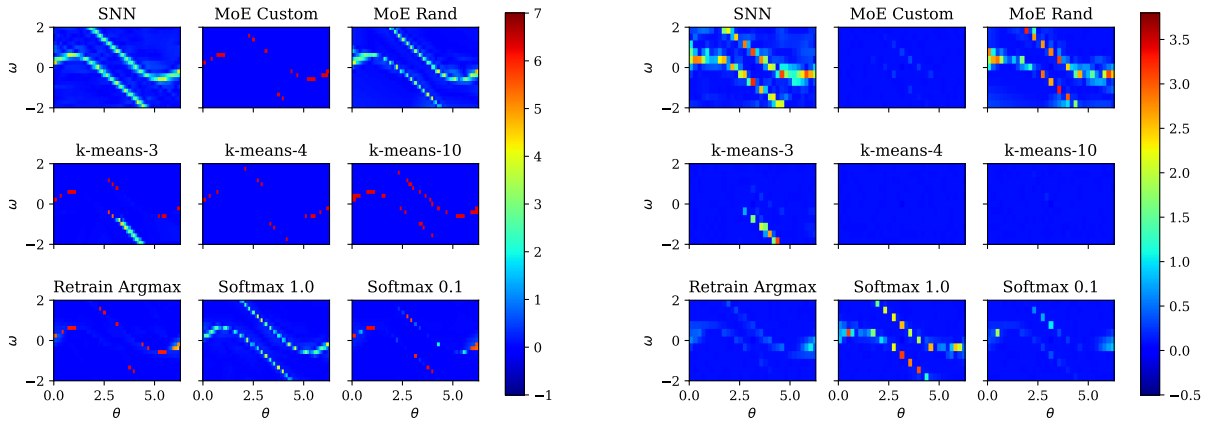
We consider a point robot navigating inside an indoor environment. The indoor environment is composed of 3 rooms and 2 doors where each door connects two neighboring rooms, as shown in Fig. 6.14. The point robot has to move from its current location p_{robot} to the goal p_{goal} while avoiding collision with the walls. Polynomials are used to represent the robot's path for each degree of freedom, denoted as $\{x(t), y(t)\}$. The cost function of a path is chosen as the jerk of the path added by a penalty on path time, i.e.

$$J = w(t_f - t_0) + \int_{t_0}^{t_f} (\dddot{x}(t)^2 + \dddot{y}(t)^2)\mathrm{d}t \tag{6.4.6}$$

with $w = 1$.

We assume the robot does not have access to its location or environment map but equipped with a distance sensor to its surrounding. Since the location of the robot is unknown, this is a partially observable problem. The target position is given as the offset from the robot to the goal, i.e. $p_{\text{goal}} - p_{\text{robot}}$. The distance sensor returns the robot's distances to the nearest obstacles in all directions within some range. The intuition is that if the environment is fixed, the sensor reading depends on the robot location and the inverse mapping from the sensor reading to the robot's location may be learned. However, the inverse mapping is not unique for some observations, so is the mapping from sensor reading to the optimal trajectory. Another source of mapping non-smoothness is sometimes the trajectory does not depend on the sensor reading in scenarios when the robot is close to the goal and a straight path is feasible. A rigorous approach to this problem is to compute optimal trajectories in the belief space. However, this is beyond the scope of this paper and we leave it as future work. The robot position is assumed known in data generation to compute the optimal trajectories. In later steps of model training and validation, the robot position is not used.

We manually construct a simple indoor environment composed of 3 rectangles connected by 2 doors, as shown in Fig. 6.14. Discontinuity arises when two close targets are at different sides of the wall, which results in similar goal offsets but quite different paths. The distance sensor resolution is chosen as 72 and a maximum range of 0.5 is used. It is used as a validation problem to test if the proposed learning pipeline based on clustering metric gives reasonable performance.



(a) Prediction error of θ_f

(b) State error after trajectory tracking

Figure 6.4: Comparing several models for learning the pendulum swing-up task. (a) shows the prediction on θ_f for a uniform grid of initial states from different models. (b) shows the state error after tracking a trajectory predicted by different models.

6.5 COMPARISON OF TRAINING APPROACH

We claim that independent pre-training of each component of MoE after careful data clustering is a superior training approach to backpropagation and joint training. In this section this claim is empirically verified on the pendulum benchmark problem.

6.5.1 Experiment Setup

The pendulum swingup problem is chosen because it has low parameter dimensionality and is ideal for visualization of results. We compare on two metrics: 1) test error (smoothed L_1 loss) and 2) rollout success rate.

The following variations are considered:

1. SNN vs MoE,
2. Joint training vs decoupled training of MoE
3. Retraining after pretraining vs no retraining.

We summarize the choices to be made when MoE is trained with different approaches. Choices such as network depth and size, optimizer, and step size are omitted since all approaches have these choices to make. The decoupled training only depends on how the data is clustered since it always uses random weight initialization and the number of experts equals

the cluster number. For decoupled training, we test the following clustering strategy: 1) customized clustering based on physical knowledge; 2) k-Means clustering with $k = 3, 4, 10$.

There are several choices to make when MoE is trained jointly:

1. Number of experts: this determines the structure of the network.
2. Weight initialization: weights are initialized randomly or from pretraining.
3. Argmax or softmax: it determines if a single or mixture of predictions is used.
4. Parameter ε: if softmax is used, this controls how much softmax approximates argmax, as in Eq. (6.3.6).

For joint training, the following settings are tested: 1) 3 experts, random initialization, argmax, $\varepsilon = 1$; 2) 3 experts, pretraining, argmax; 3) 3 experts, pretraining, softmax, $\varepsilon = 1$; and 4) 3 experts, pretraining, softmax, $\varepsilon = 1$. We note that 3 experts are chosen to match the decoupled training setting. The loss function to minimize is Eq. (6.3.5) since it is a regression problem.

As a baseline, the SNN is chosen of size (2, 300, 75), where the first number denotes the size of the input layer, the last number denotes the size of the output layer, and intermediate numbers indicate the size of hidden layers. We experimented with SNN with more hidden layers or more neurons in the hidden layer, but they result in similar or larger test error. For MoE, the classifier is of size (2, 50, r) and the r regressors are all of size (2, 20, 75).

In all the experiments, hidden layers use LeakyReLU activation function with $\alpha = 0.2$. The output layer of regressors is linear. The input and output for each model are preprocessed by standardization. Smooth L1 and cross entropy loss are used for regression and classification, respectively. Neural network training is the same for both SNN and MoE, where 80% and 20% data are used as training and test set, respectively. Adam [142] is used to train all neural networks with default parameters. The learning rate is always set to 0.001. Training is stopped when test error does not decrease within the last 10 epochs. Additional validation set is sampled for trajectory rollout from the same distribution.

6.5.2 Results

Fig. 6.4.a plots the prediction error on θ_f and Fig. 6.4.b plots the state error after trajectory tracking. The validation error and rollout success for each model are also listed in Tab. 6.2.

Row 1 shows that SNN makes poor predictions in regions near the discontinuity, averaging between both sides of the boundary, causing moderate prediction error in θ_f, and having

Table 6.2: Comparison of prediction error and rollout success rate on the pendulum problem

Model	SNN	MoE							
Clustering	—	Custom	Rand.	k-Means-3	k-Means-4	k-Means-10	Custom	Custom	Custom
Retrain	—	—	—	—	—	—	argmax	softmax 1.0	softmax 0.1
Validation error	0.046	0.030	0.035	0.039	0.029	0.051	0.027	0.028	0.026
Success (out of 1000)	717	998	829	970	1000	1000	941	896	969

large tracking error. MoE with custom clustering does also make large prediction error in θ_f for states near the discontinuity, but these are caused by misclassification and the prediction is a local optimal trajectory belonging to another cluster. Hence, they are suboptimal but still reach the vertical position as desired, since the difference in θ_f is 2π. The suboptimality is not too great, because near the boundaries two different local optimal trajectories have similar costs. MoE trained from random initialization does achieve lower prediction error than SNN, but is not very successful at predicting θ_f or trajectory tracking. This indicates that training MoE by minimizing Eq. (6.3.5) is unable to identify the discontinuity.

Row 2 tests MoE with k-Means clustering of different k, which are shown in Fig. 6.5. Assuming the custom clustering as the ground truth, $k = 4$ and $k = 10$ finds the discontinuity successfully with possible over-segmentation, but $k = 3$ only partially clusters the data. The tracking success rate shows over-segmentation does not degrade tracking performance, although it does result in higher classification error (as it has more red pixels in Fig. 6.4).

Row 3 of Fig. 6.4 shows results of retraining after pretraining MoE with custom clustering. In all cases this approach decreases regression error but also rollout success rate. If argmax is used, the classifier has no gradient to update itself so only the regressors are updated. Due to classification error, the regressors will be trained with trajectories from other clusters. As a result, the predictions near the classification boundaries will tend towards the average of two clusters. If softmax is used, the classifier is updated but the output is a weighted sum of predictions from all experts, and thus has the same averaging issue. As shown in Tab. 6.2, retraining does decrease the prediction error at the cost of lower rollout success rate.

6.5.3 Discussion

These experiments suggest that proper clustering is important for MoE training. Backpropagation is unable to find an appropriate clustering from randomly initialized classifier or classifier trained with incorrect clustering. Moreover, rollout success is a better metric to use in practice, while testing error can be misleading. Due to misclassifications, a lower testing error can be achieved by averaging at discontinuities, but this leads to severe failures.

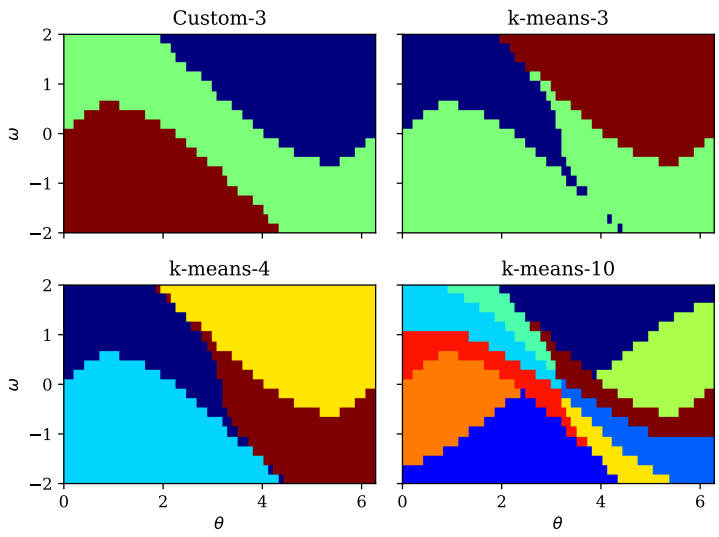


Figure 6.5: Choices of clusters for the pendulum problem. Different colors means different clusters. Figures include: 1) custom 3 clusters 2) k-Means with 3 clusters 3) k-Means with 4 clusters 4) k-Means with 10 clusters

We also observe that coupled retraining is detrimental to performance. This is because the imperfect classification causes the individual regressors to be trained with data from other regions, again leading to averaging artifacts.

The rationale of retraining is that pretraining provides a good initialization, but if the data is clustered badly, i.e. in one cluster there is discontinuity, the loss function may be large. The hope of retraining is to use backpropagation to update the classifier to find the correct clustering assignments if the initial assignment in pretraining is not. However, we show that retraining typically reduces model prediction error, but also decreases rollout success rate. More specifically, if the initial clustering assignment has discontinuity, retraining might improve but is unlikely to find the correct clustering. Even if the initial clustering is perfect, retraining sacrifies rollout success for prediction error.

In order to obtain high trajectory tracking success rate, we have to 1) use *argmax* to select one regressor; and 2) avoid training regressors with data from more than one clusters. This raises the problem of how to appropriately cluster the trajectories which is studied in the next section.

6.6 TRAJECTORY CLUSTERING APPROACH

In the previous section, we show that the performance of MoE depends on the clustering assignment and we cannot rely on coupled training to find an appropriate clustering.

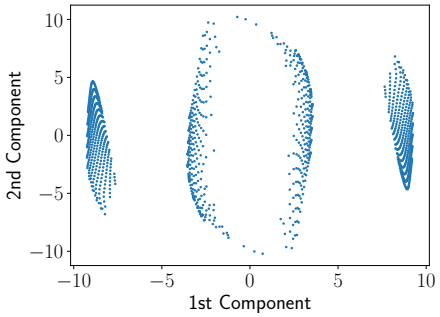
Therefore it is worthwhile to investigate a general and scalable clustering approach that is applicable to a wide spectrum of problems since expert knowledge is not always available. Moreover, it is beneficial to have visualization tools to gain insights of the data structure and help to tune hyperparameters. In this section, we experimentally explore how to visualize the dataset. Moreover, several metrics are proposed in order to predict if a clustering leads to reasonable data segmentation.

6.6.1 Trajectory Visualization

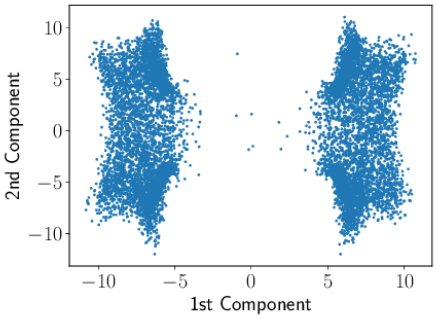
The trajectory is represented by a high-dimensional vector. In order to visualize such high dimensional data, we experimentally explore a few approaches for trajectory visualization to gain insights of the data structure.

PCA PCA is a simple yet effective and efficient approach for dimension reduction and data visualization. It decomposes the data by projecting onto orthogonal directions sorted by variance along those directions. We draw in Fig. 6.6 the scatter plot of the first two components for the first 4 benchmark problems. It should be noted that the first two major components might not explain sufficient variance in the data and two components are chosen only for visualization purpose. As Fig. 6.6 shows, PCA provides useful information on the existence of disconnected components for both pendulum and ground vehicle problems. However, for the two drone problems, no additional information is obtained. Moreover, the PCA plot shows the pendulum problem has 4 clusters, which is different from human intuition. The PCA plot for ground vehicle could be misleading since it only shows existence of 2 clusters. However, other technique suggests the existence of more than 2 clusters.

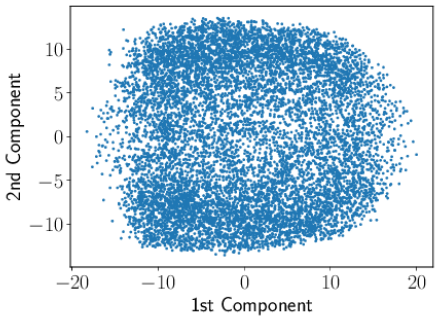
UMAP There is extensive research in nonlinear dimension reduction for data visualization such as t-SNE [143] and UMAP [144]. Here UMAP is used due to its advantage of computational efficiency over t-SNE [144]. However, computational efficiency limits us to use at most 100,000 downsampled data for the two drone problems. All the results are obtained using the default setting and shown in Fig. 6.7. As the figure shows, UMAP constructs low-dimensional embeddings as 4 and 6 disconnected components for the pendulum and ground vehicle problems, respectively. The 4 components structure agrees with PCA results for pendulum problem. But UMAP extracts more structures for ground vehicle trajectories than PCA. For the drone problems, the trajectories are actually not in disconnected components so UMAP is unable to embed the original data into several disconnected components.



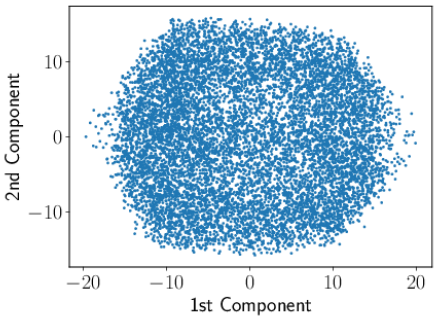
(a) Pendulum



(b) Ground vehicle

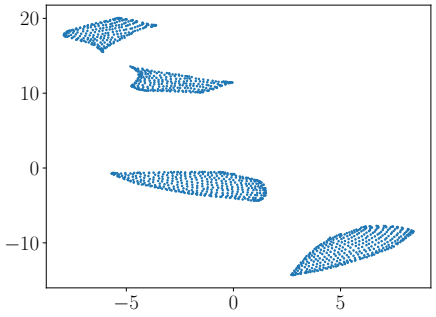


(c) Drone-obstacle

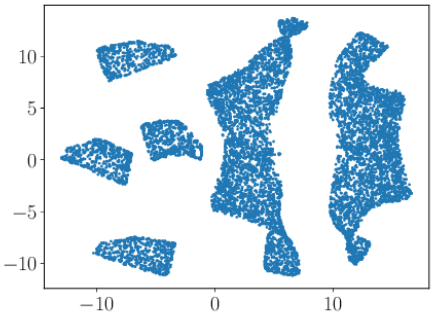


(d) Drone-two-obstacles

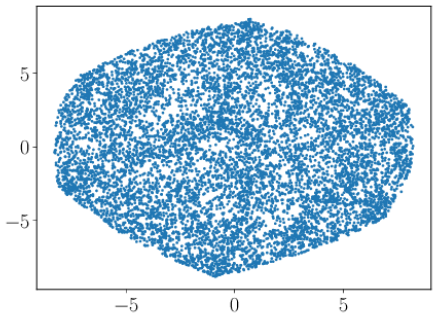
Figure 6.6: First two major PCA components for benchmark problems.



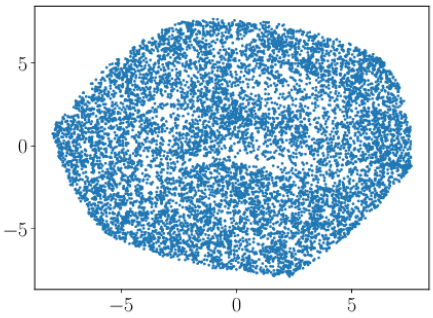
(a) Pendulum



(b) Ground vehicle



(c) Drone-obstacle



(d) Drone-two-obstacles

Figure 6.7: Nonlinear manifold projection for benchmark problems.

Summary We suggest two approaches for trajectory visualization to gain insights of the trajectory structures. These two approaches are both dimensionality reduction method to project the trajectories onto a plane for visualization. Structures like disconnected components are revealed by these approaches. Both approaches are good at showing the existence of disconnected components, but PCA might result in incomplete or misleading results. Neither approach provides insights on the two drone problems due to its different discontinuity pattern. We suggest using these techniques to detect the existence of disconnected components.

6.6.2 Clustering Metric

A suitable clustering metric provides a surrogate to the performance of MoE without training due to the close connection between MoE performance and data segmentation quality. There exist many clustering metrics in literature such as those studied in [145] but they do not necessarily reveal intra-cluster discontinuity which may lead to MoE failure. The clustering metric has to be able to reveal discontinuities within a data subset which is essential to MoE performance. Moreover, although we can define discontinuity between any two data by path connectedness, it is computationally prohibitive to verify connectedness for every pair in a large dataset. It is reasonable to only consider function discontinuity in a local region by the definition of discontinuity. Therefore for one example (p_i, z_i) we find the neighbors of p_i using nearest-neighbor methods in the parameter space and study the corresponding trajectories. This is performed for each example in a data subset to get an overall evaluation. We note that the performance of our approach is: 1) directly related with intra-cluster discontinuity and 2) not sensitive to over-segmentation. The metrics are designed to focus on function discontinuity. There exists many cluster validity indices to evaluate and compare performances [145]. Our clustering of trajectories is closed related with image segmentation [146] where it also tries to find regions that are spatially close but far in pixels. Most of the cluster indices estimate the cluster cohesion (intra-distance) and cluster separation (inter-distance) and combine them to compute a quality measure [145]. However, the intra-distance is usually calculated by the average distance from all data to the cluster centroid and is thus perferable to spherical shape while in fact the metric should be shape-invariant. Besides, the intra-distance depends on overall data distribution and does not necessarily reveal function discontinuity which depends on local information. As a result, the conventional cluster indices may not work well. Our data is also fundamentally different from images and the image segmentation indices [146] cannot be directly used. For example, the metrics based on pixel error or texture are not applicable. Besides, for the high

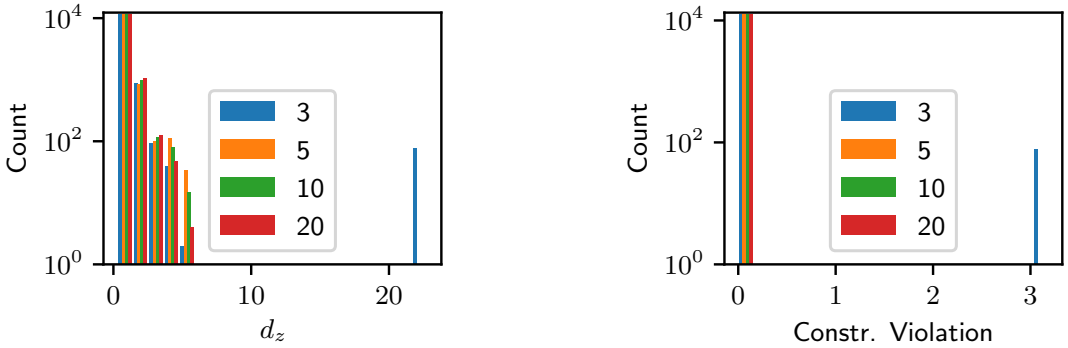
dimensional trajectory data many concepts in images do not have alternatives. Our metric is more like trying to find segmentations such that within each region no edge exists.

Metric proposal We propose the following three metrics as surrogates of function discontinuity. These metrics are computationally cheap, based on pair-wise computation, and easily visualized through histograms. A comparison with several existing metrics are shown later. For a dataset of n data and choice of m neighbors, a surrogate function $f(\cdot,\cdot,\cdot,\cdot) \in \mathbb{R}^l \times \mathbb{R}^R \times \mathbb{R}^l \times \mathbb{R}^R \to \mathbb{R}$ is evaluated for each example (p_i, z_i) and each of its neighbors $\{(p_j, z_j)\}_{j\in\text{Neighbor}(i)}$. By evaluating the metric function for every data and its neighbors, a histogram can be used to summarize the overall segmentation quality. We can further convert the histogram into a scalar for comparison among cluster numbers. The *inflection point* can be used to choose the optimal cluster number. Surrogate functions include:

Trajectory distance: Function discontinuity usually means small distance in p but large distance in z, so trajectory distance is a promising surrogate, i.e. $f(p_i, z_i, p_j, z_j) = d_z(z_i, z_j)$ for some distance function d_z such as 2-norm. The assumption is p is sampled densely so trajectory distance between neighbors is small unless discontinuity exists. An overall distribution of trajectory distance between neighbors thus indicates the probability of function discontinuity within a data cluster. However, this approach might not perform well when data is so sparse that even neighbors from a continuous region have large distances. An alternative is the ratio of trajectory and parameter distances. For a multivariate function, a surrogate of gradient is $f = d_z(z_i, z_j)/d_p(p_i, p_j)$ for distance function d_z and d_p, which approximates the norm of the Jacobian. A larger ratio indicates a higher chance of discontinuity. However, this approach involves division of $d_p(p_i, p_j)$ and is susceptible to trajectory noises if two samples are close. Similarly, the trajectory distance $d_z(z_i, z_j)$ reveals discontinuity if two close parameters have large trajectory distance. The trajectory distance is more robust to data noise but may not perform well when data is sparse where neighbors from a continuous region have large distances. The distance ratio and trajectory distance metrics perform similarly in our cases and we only present results from the trajectory distance metric.

Homotopy class check: Continuity leads to the existence of homotopic trajectories to the neighbors. An indicator of function discontinuity is whether the straight line connecting two data belongs to the dataset. The surrogate is thus $f = c((p_i + p_j)/2, (z_i + z_j)/2)$ for some constraint violation function $c(\cdot,\cdot) \in \mathbb{R}^l \times \mathbb{R}^R \to \mathbb{R}$. We note that this only checks the *midpoint* between p_i and p_j. Besides, we only evaluate the violation of constraints instead of solving $z((p_i + p_j)/2)$ for efficiency reasons. This resembles connectedness check and it is

able to detect the discontinuity type appearing in the shortest path problems.



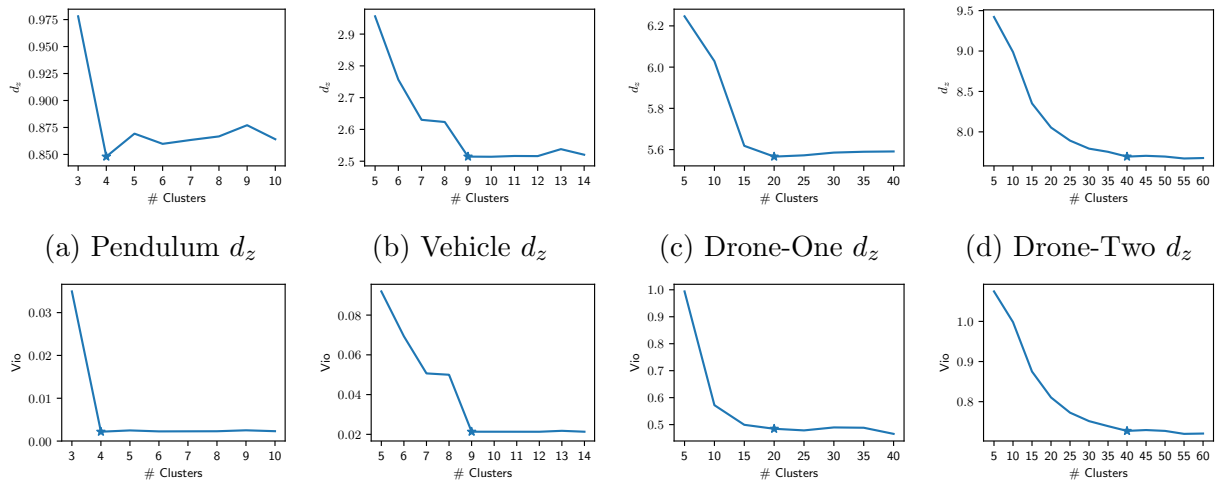
(a) d_z (b) Constr. Vio.

Figure 6.8: Histogram of metrics for PCA+k-Means for the Pendulum task. Colors denote choices of cluster numbers. All figures show 3 clusters are not sufficient to cluster the dataset well, as the distinctive blue bar on the right shows. As k increases, the histograms are moving left, showing better function continuity property.

Comparison of Metrics We show the histogram of metric functions with different cluster numbers for the pendulum problem in Fig. 6.8. The existence of bins at the right clearly shows trajectory discontinuity when $k = 3$, and there does not exist discontinuity within each cluster for other k. A scalar value for each histogram is calculated and shown in column one of Fig. 6.9. The inflection point clearly shows $k = 4$ is the optimal cluster number if a finer grid is used. In practice, a coarse grid is sufficient to determine a satisfactory k.

Fig. 6.9 also shows the metric values for other problems. If the inflection point is chosen, the optimal clsuter numbers for the benchmark problems are 4, 10, 20, and 40, respectively. As shown later, this agrees with the rollout results in Fig. 6.10. However, we note that the constraint violation metric for the two drone problems is not quite meaningful due to the small change in values.

Summary We propose 3 metrics as indicators of function discontinuity. All metrics can be used to compare the amount of discontinuity between clustering assignments. However, constraint violation serves as better metric since it has clear unit and is easier to set a threshold. The d_z/d_p metric is a strong indicator of discontinuity, but it might be susceptible to trajectory noises. Moreover, the threshold is rather difficult to choose since its range of values may be large. The difficulty in threshold selection also applies to d_z metric. Although these metrics are not sufficient to determine whether a dataset is continuous or not, they can be used to compare between different assignments.



(a) Pendulum d_z (b) Vehicle d_z (c) Drone-One d_z (d) Drone-Two d_z

(e) Pendulum Vio. (f) Vehicle Vio. (g) Drone-One Vio. (h) Drone-Two Vio.

Figure 6.9: Comparison of metric values for different cluster numbers for each benchmark problem. We note that in subplot (g) and (h) the metric values changes so little that they are less meaningful. We note that a fine grid is used here to illustrate the metrics and in practice we test performance on a grid showing in Tab. 6.3.

Comparison with cluster validity indices There exist many clustering metrics that are used to evaluate clustering algorithms. We refer [145] for an extensive comparison of some. We select 3 metrics studied in [145] that perform well for the datasets used in [145]. The metrics include: Calinski-Harabasz index (CH), Silhouette index (Sil), and Davies-Bouldin index (DB). We compute them by subroutines provided by scikit-learn [147]. Since we perform clustering after dimensionality reduction, these metrics are also computed using the data after PCA. All metrics are functions of separation which measures how different clusters are from each other and cohesion which measures how similar data in the same cluster are. Higher values in the first two and lower value in the last metric indicates better clustering quality. The results are in Tab. 6.3. The bold entries indicate the best clustering according to the chosen metric. CH tends to choose values at the boundaries and are thus not favorable. Sil and DB works well for the first two problems but fails for the last two. The results clearly show existing clustering metrics are not well suited for our problems. This is not beyond expectation since these metrics are shape dependent and do not reveal intra-cluster discontinuity.

Table 6.3: Evaluation of several existing clustering metrics clustering metrics in the literature. Bold indicates # clusters suggested by metric, which fail to correlate with performance in Fig. 6.10.

	Pen				Car				DroneOne					DroneTwo				
k	3	5	10	20	5	10	20	40	5	10	20	40	80	5	10	20	40	80
CH	1.52e3	1.72e3	2.40e3	**2.90e3**	**3.92e4**	3.81e4	3.04e4	2.43e4	**4.42e4**	3.49e4	2.47e4	1.82e4	1.30e4	**8.74e4**	6.74e4	5.12e4	3.87e4	2.76e4
Sil	0.508	**0.539**	0.484	0.399	0.290	**0.332**	0.326	0.261	**0.205**	0.183	0.168	0.163	0.153	**0.175**	0.164	0.171	0.162	0.152
DB	0.844	0.760	**0.692**	0.788	1.397	**1.067**	1.184	1.218	**1.385**	1.511	1.559	1.425	1.407	1.601	1.602	1.507	1.414	**1.396**

6.7 TRAJECTORY ROLLOUT RESULTS

Trajectory rollout is the test metric for model evaluation since it directly reveals the performance of a deployed model. We compare SNN, and MoE with different cluster sizes on this metric. The model size of SNN is fine-tuned and shown in Tab. 6.1 by enumerating a few network depth and hidden layer size combinations to achieve the lowest test error, so SNN has some advantage. The MoE is trained individually according to data clustering and the regressors use the same depth with SNN but the hidden layer sizes are chosen such that

1. The number of parameters of each regressor is proportional to the data size of the cluster.

2. The sum of regressors parameter number equals SNN.

The rollout result is computed on a validation set and the regression loss is neglected. The rollout metric for the pendulum and vehicle problem is the success rate while the two drone problems use constraint violation. In Fig. 6.10 a result summary is shown with the effect of k. MoE outperforms SNN in all benchmark problems. As the cluster number increases, MoE performance increases and eventually decreases slowly. The clustering metrics agrees with the rollout performances.

6.7.1 Examples of SNN vs MoE

To demonstrate why SNN tends to fail near function discontinuity while MoE is able to handle this, we further show a few examples for the benchmark problems. In Fig. 6.11 we show the predictions from SNN and MoE on a selected parameter as well as the optimal trajectories of its neighbors for the vehicle problem. Similar to the pendulum problem, SNN may fail to predict θ_f correctly.

Fig. 6.12 shows examples of optimal trajectories and predictions from SNN and MoE for the drone problem with one obstacle. As the initial state moves along z direction, the optimal trajectories change from going above to going below the obstacle. SNN is unable

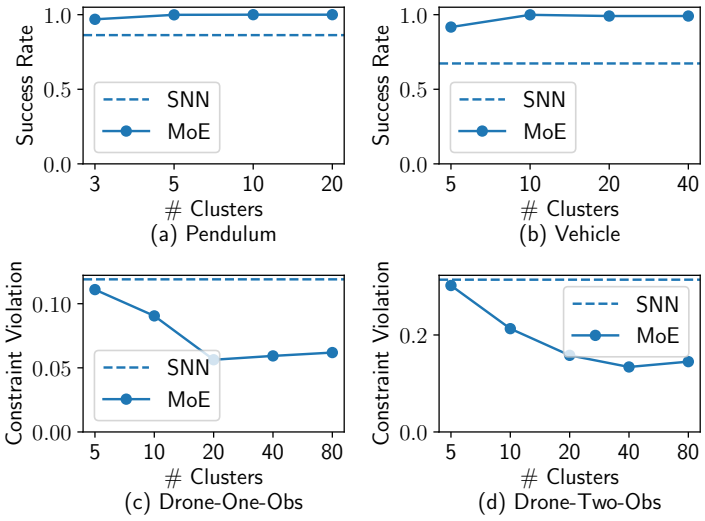


Figure 6.10: Comparison between MoE and SNN on benchmark problems. MoE outperforms SNN in all problems. For (a) and (b), MoE achieves higher than 99% success rate. For (c) and (d) SNN has 50% improvement in terms of constraint violation. It also shows as k increases, MoE performance increases and eventually decreases.

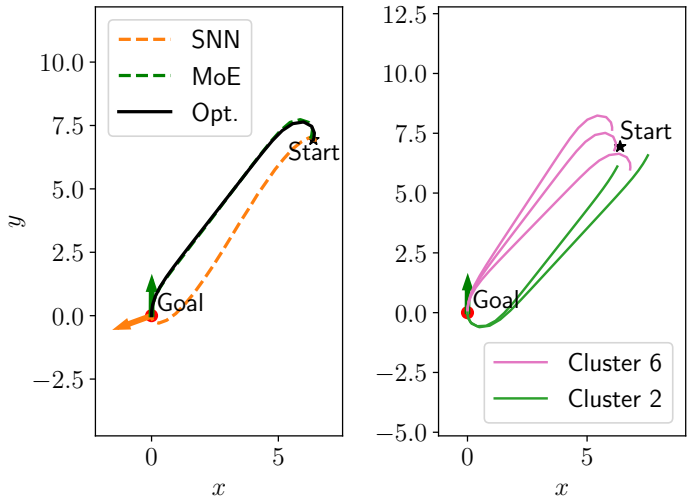


Figure 6.11: Left: The optimal trajectory and predictions (no tracking) from SNN and MoE for a chosen start state for the Vehicle problem. MoE makes accurate prediction while SNN predicts an average trajectory of two clusters. The arrow denotes the final angle (straight up is desired) and SNN makes large prediction on it. Right: the neighbors of the chosen state. The pink and green lines are the neighbors of the start so they start from different locations. They demonstrate the existing discontinuity within trajectories.

to handle such discontinuity and predicts a trajectory that violates the obstacle avoidance constraint. However, MoE is able to reason about the discontinuity and predicts trajectories using different experts.

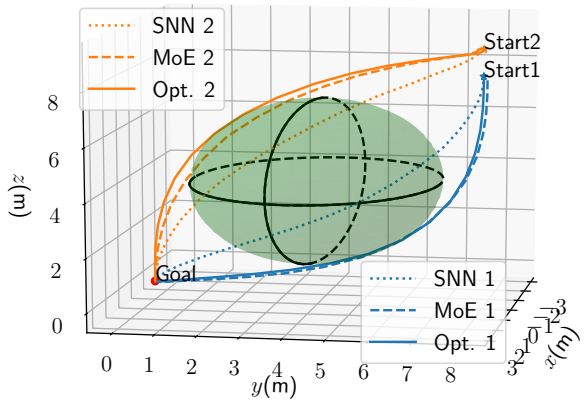


Figure 6.12: Optimal trajectories and prediction (no tracking) from SNN and MoE for two selected close states for the Drone-One-Obstacle problem. SNN predicts a trajectory that violates obstacles avoidance constraints. Green sphere: obstacle with center at (0, 4, 4) and radius 3. Solid, dashed and dotted lines: optimal trajectories, prediction of MoE, and prediction of SNN, respectively.

6.7.2 Test of Learning Pipeline on Navigation Problem

The indoor navigation problem is used as a test problem on the trajectory learning pipeline proposed in this paper. Specifically, we want to test if the clustering metrics help to choose a clustering assignment and how MoE performs based on this choice. We note that this an expert can manually divide the dataset of this problem into 9 clusters based on the room numbers of the start and goal. This manual clustering serves as a baseline. For metric computation, PCA is used to reduce dimensionality to 6 (resulting in 90% explained variance) before Nearest Neighbor algorithm is used. The trajectory distance-based metrics are shown in Fig. 6.13 and indicates 20 clusters is the best choice among the three choices. The constraint violation metric is chosen as whether the midpoint has collision with the environment and listed in Tab. 6.4.

The metrics indicate $k = 20$ might result in best performance and we directly fit MoE using clustering assignemnt by $k = 20$. For a test set of 1000 examples, MoE results in 9 collisions while SNN gets 36 collisions. The reduction in collisions shows MoE improves upon

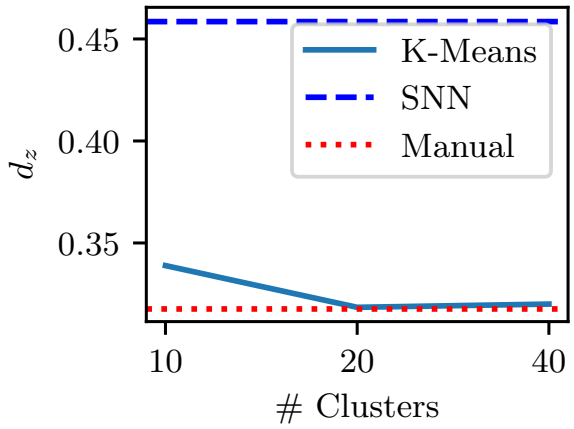


Figure 6.13: Comparison of the trajectory distance metric for the Navigation task. It shows $k = 20$ is a reasonable choice among the 3. The metric value is large if no clustering is performed and small for a manual clustering.

Table 6.4: Number of pairs whose midpoint has collision for several clustering assignments

SNN	Manual	$k = 10$	$k = 20$	$k = 40$
32722	30	9978	1059	4310

SNN near discontinuity. Note that for different problems, the region of discontinuity may be of different sizes and the improvement may not be great if the region is small. However, MoE is still necessary to achieve reliable success rate to minimize requirement of human intervention and prevent catastrophic failures.

Fig. 6.14 shows examples of optimal trajectories and predictions from SNN and MoE. Similar to the drone problem, SNN predicts averages between trajectories and leads to collision.

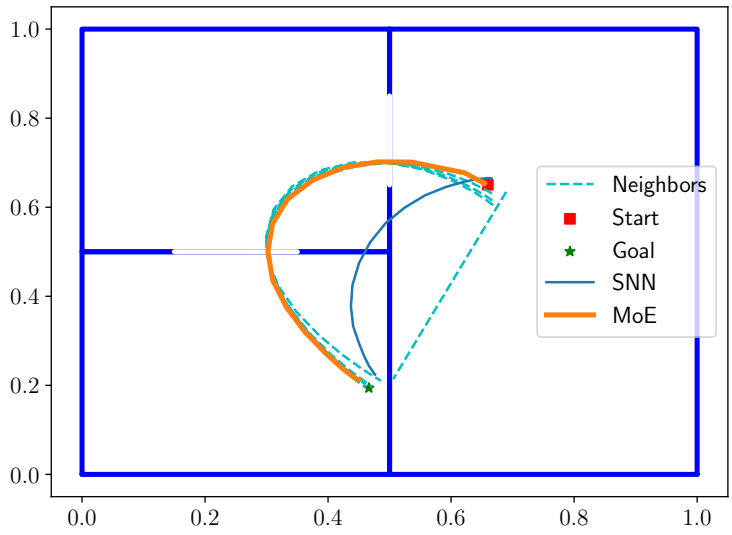


Figure 6.14: Comparison of SNN and MoE on a selected example. The dashed lines are the neighbors of this example. The blue and orange lines denote prediction by SNN and MoE, respectively. The neighbors showing existence of function discontinuity and MoE is able to handle it while SNN predicts a path with collision.

6.8 CONCLUSION

In this chapter we demonstrate that optimal trajectories can be learned with high accuracy if we take into account the special structure of optimal control problems. MoE is designed such that each expert approximates a smooth region in the problem-optimum mapping, and the classifier handles discontinuities without averaging. It is important to train MoE with the correct clusters, and curiously, coupled training of the regressors and classifier tends to be detrimental to tracking performance. We also argue that test error is not a good metric to judge learning models, but rather rollout success rate under trajectory tracking is preferable. Several metrics are proposed to evaluate existence of discontinuity and compare between clustering assignments. These metrics are proved useful in selecting cluster numbers. For

a typical PCA+k-Means clustering pipeline, we find that as the cluster number increases, the MoE performance tends to increase first and eventually decrease. Deep RL with policy gradient algorithm also suffers from policy discontinuity issue and tends to fail as problem becomes more complex.

One limitation of this approach is some problems are not easily parameterizable.si Take the drone with obstacle problems as an example, if we further increase obstacle numbers, the data collection and training procedure have to be repeated. Moreover, for obstacles with irregular shape, it is not easy to find a low-dimensional representation. One approach is to use some redundant parameterization such as scenarios images. Another limitation is how to determine whether the collected data is sufficient or not.

Future work includes developing more sophisticated clustering algorithms that automatically find the best clustering assignment. Solving the discontinuity issue in RL is another problem to investigate. Further work also includes how to prove the stability of trajectory rollout using the predicted trajectories.

CHAPTER 7: DISCONTINUOUS FUNCTION LEARNING WITH MIXTURE OF EXPERTS AND TOPOLOGICAL DATA ANALYSIS

In Chapter 6 Mixture-of-Experts (MoE) models are used to handle the discontinuities in the optimal solution for parametric planning tasks. In that context, the existence of discontinuity is found with domain knowledge. Training of the experts requires us to split the data into groups, which is done by k-Means clustering with a carefully tuned number of clusters. Despite the empirical success of clustering, there is no guarantee of its performance. In this chapter, we present an approach with more theoretical guarantees of performance that computes topological features from data, which are, then, used to split the dataset into continuous pieces. We provide a proof that shows any change to the computed topological features is a sufficient condition for the existence of function discontinuities in the region. Three types of changes of topological features are studied and different data split approaches are proposed for each type. For the discontinuity type caused by geometric singularity in the 3D obstacle avoidance context, the topology-based approach outperforms the k-Means method presented in Chapter 6. [1]

[1] Ongoing work.

7.1 INTRODUCTION

Training of MoE to approximate discontinuous function requires us to split the training data into groups within the domain of which the objective function is continuous. The k-Means method in Chapter 6 uses clustering of trajectories after dimensionality reduction with PCA to approximate the data split and requires fine-tuning of the hyperparemeter k. The reasons why fine-tuning k is necessary are:

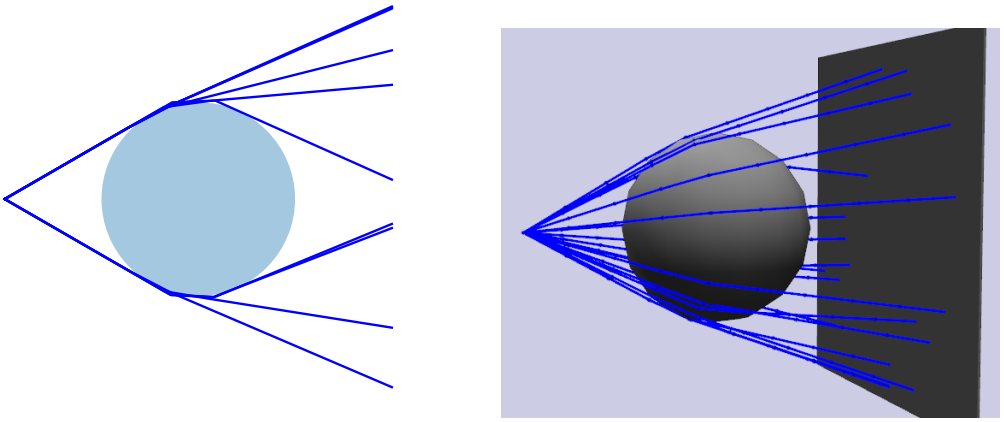
1. There is no *a-priori* information on how many groups are necessary;
2. There are no theoretical guarantees that k-Means is capable of removing all discontinuities in this task with a reasonable k.

Since k-Means is based on Euclidean distance which may perform poorly for trajectory data that lies on some manifold, the results in Chapter 6 show k-Means clustering usually requires over-segmentation to get decent results. Although over-segmentation is not as serious an issue as under-segmentation, the results in Chapter 6 show that the model prediction becomes worse as the number of clusters grows too large since the available data for each expert becomes scarcer. This motivates the search for a better data split approach in order to avoid the issues from both under and over segmentation.

One noticeable difference between the 2D and 3D obstacle avoidance problems in Fig. 7.1 is that all the trajectories of the 3D problem can be continuously deformed into one another while for 2D the trajectories above the obstacle cannot be deformed into those under the obstacle. Moreover, a cycle can be found in the trajectories of the 3D problem that cannot be continuously deformed into a single trajectory while such a cycle cannot be found in the 2D problem. add an example of cycles... The differences in these behaviors are caused by the different topology of the trajectories. A good approach to split the trajectories of 2D problem is to split them according to whether they are above or below the obstacle, but this approach is not sufficient for the 3D case. Considering the smooth transformation between trajectories to be an indicator of continuity, we aim in this chapter to consider the topology of the dataset and use it as a basis to perform the dataset split.

Note, however,that, in this example, domain knowledge is used to identify the difference in topology, which limits the generalization of this method to other problems. Fortunately, Topological Data Analysis (TDA) [148] is a computational tool to find the topological structures from data. With the help of TDA, domain knowledge is not required to figure out the topology of the space from which a dataset is sampled. It is able to compute the topological features from a dataset such as the zeroth homology group (H_0, connected components) and the first homology group (H_1, 1D hole). Higher order homology groups are often too

expensive to compute and are thus not investigated here. Specifically, it helps to compute the number of connected components and 1D holes. For instance, the 2D problem has 2 connected components for the trajectories and 0 1D hole, while the 3D problem has 1 connected component and 1 1D hole. All of these properties can be computed by TDA. Later we show that these features from TDA help determine the existence and type of discontinuities of the argmin function. The information of discontinuity type is used to determine how to split the dataset for better MoE training. Examples are shown to demonstrate the efficacy of these data split methods.



(a) 2D (b) 3D

Figure 7.1: Samples of trajectories for the 2D and 3D obstacle avoidance problem. These two problems have different topological structures.

The contribution of our work includes:

1. Prove sufficient conditions for function discontinuity with changes in computed topological features.
2. Propose corresponding methods to split the data for different changes of the number of connected components and 1D holes in order to improve MoE.
3. A topology-based approach to split data with one type of geometric singularity commonly seen in 3D obstacle avoidance problems.

7.2 RELATED WORK

Trajectory learning has shown advantages over approaches that compromise optimality for real-time efficiency, as Chapter 5 shows. Blindly learning with continuous SNN underperforms MoE in discontinuous function approximation, as shown in Chapter 6. However,

successful training of MoE requires the data to be split into continuous pieces, which is non-trivial without domain knowledge. The k-Means method in Chapter 6 uses clustering to approximately find a split of data, but has no theoretical guarantees of success or distinguish discontinuity types. Moreover, empirically it needs careful tuning of k, and fails to identify some discontinuities. In this chapter, topological structures of the data are used to identify function discontinuity with the help of TDA.

In motion planning, the topology of the environment and trajectories are both used. In [148] the homotopy class of the trajectories is used to help with graph search-based motion planning. It can find paths respecting given homology class constraints and explore different homotopy classes in an environment. Topology is also used in [149] to deal with system uncertainty. In [150] topology is used to study the complexity of motion planning algorithms in navigation tasks. These methods, however, rely on graph search methods to compute the optimal plan and do not consider learning from trajectories. The most related work [72] also uses TDA to extract multimodality information from precomputed trajectories, cluster them and train MoE models using these clusters. However, they are not considering H_1 structures in the trajectory data, which occur frequently in robotics applications.

7.3 PRELIMINARY

In this section, we briefly introduce several essential components to help the readers understand the terminologies throughout this chapter with the help of examples.

We refer to readers interested in the details of computational topology to [151] for a thorough exploration of the topic.

7.3.1 Betti Numbers

The topology of a space describes how all the points in the space are connected [151] and it is invariant to many types of continuous deformations. For instance, a closed curve is different from an open curve since it divides the plane into 2 disjoint parts and it has no ends. A circle and square are topologically equivalent since they are both closed curves and have the same space division property. A cylinder is different from a donut as the donut has a hole while the cylinder doesn't. As a result, a closed curve that wraps around the donut's smaller ring cannot be continuously shrunk into a point while such curve does not exist for cylinder. Computational topology studies the topology of a topological space using sampled data. For instance, if the data is sampled from an annulus, computational topology is able to establish its topological equivalence to a closed curve.

Homology is a way of associating algebraic groups to a topological space with the same topology. For instance, an annulus curve is associated with S^1 group since they share the same topology. A disk, however, is associated with the zero group (a group consisting of a single element). Widely used groups in computational topology are the zeroth dimensional homology group (H_0) which relates to the number of connected components in the structure and the first dimensional homology group (H_1) which is related to the number of one-dimensional holes. The ranks of H_0 and H_1 are called the zeroth and first Betti numbers, denoted as b_0, b_1, representing the number of connected components and 1D holes, respectively. A closed curve, like the annulus, has $b_0 = 1, b_1 = 1$ and an open curve, like a line segment, has $b_0 = 1, b_1 = 0$.

7.3.2 Cycles and Homotopy

A continuous function $p : [0, 1] \to C$ from interval $[0, 1]$ to a subset C of a topological space X is called a *path* on X between $p(0)$ and $p(1)$. A path with the same start and goal, i.e. $p(0) = p(1)$ is called a *cycle*. The degenerate case where $p(t) = p(0), \forall t \in [0, 1]$ is called a *point cycle*.

A homotopy between two continuous functions f and g is a continuous function $H : X \times [0, 1] \to Y$ from the product of a topological space X with the closed unit interval to a topological space Y such that for each point $x \in X, H(x, 0) = f(x)$ and $H(x, 1) = g(x)$. If there exists a homotopy between them, f and g are said to be homotopic. These definitions allow us to check if two cycles in the same topological space are homotopic. If one cycle can be continuously deformed into another one (i.e transformed by a continuous function H, these two cycles are said to be homotopic. The definition of cycle and homotopy helps distinguish spaces with $b_1 \neq 0$ and $b_1 = 0$. For topological space with $b_1 = 0$, any cycle is homotopic to a point cycle. However, in spaces with $b_1 > 0$, there are at least b_1 cycles that are not homotopic to a point cycle or one another. In this thesis, a cycle is called *non-contractible* if it is not homotopic to a point cycle. For space with $b_1 = 0$, there are no non-contractible cycles.

7.3.3 Persistent Homology

In order to compute Betti numbers, one has to start with building simplicial complexes from data. A k-dimensional simplex $\sigma = [x_0, \ldots, x_k]$ is the convex hull of $k + 1$ affinely independent points. Simplices of the first few dimensions have special names: *vertex* for 0-simplex, *edge* for 1-simplex, *triangle* for 2-simplex, and *tetrahedron* for 3-simplex. The

face of a simplex is the convex hull of a non-empty subset of the vertices of σ. A simplicial complex is a collection of simplices such that every face of a simplex in the complex is also in the complex and the intersection between two simplices is either empty or contained in the complex (see [148] for more rigorous definitions).

A widely used complex is the *Vietoris-Rips complex* (VR complex) which is similar to an r-neighboring graph. An r-neighboring graph of a dataset treats each data point as a vertex and connects every pair of vertices with distance below or equal to r with an edge. For a set of points in metric space equipped with a distance function, the VR complex is the set of all simplices whose maximum vertex distance is not greater than r. Clearly, with different values of r, the VR complex is different and may exhibit different topological structures, i.e. homology groups. For example, when r is very small, the VR complex is formed by isolated 0-simplices, i.e. vertices; when r is very large, the complex is composed of a high-dimensional simplex and all the faces of this simplex. Persistent homology studies the evolution of homology groups in the VR complex as r increases. As r increases, new simplices are formed and homology groups may appear (also called a "birth") and be destroyed (also called a "death"). For instance, the addition of an edge between two previously disjoint components leads to the merging of them into one component, leading the number of connected components to decrease by 1 - which can also be seen as the "death" of one of the initial components as it was absorbed into the other.

Similarly, a new edge may form a cycle with existing edges and lead to birth of a 1D hole. For each homology group, its connection radius r at its birth and death, called birth and death index, respectively, can be computed. The lifetime of a homology group, defined as the difference between death and birth index can be computed and compared. Homology groups with longer lifetimes, i.e. persistent, are more likely to be caused by the topology of the underlying space. One may plot a *persistence diagram* [151], showing the birth and death index of each homology group and the user may select groups with long lifetimes and compute Betti numbers by counting them. As an example, we show the results for applying TDA on 2D data sampled on an annulus (shown in Fig. 7.2) in Fig. 7.3. The figure indicates that the annulus has one connected component and one 1D hole so $b_0 = b_1 = 1$. The persistence diagram shows one H_0 feature and one H_1 feature have significantly longer lifetime than others so we can read from the diagram that $b_0 = b_1 = 1$. In this way, persistent homology is used to extract the topological features from sampled data.

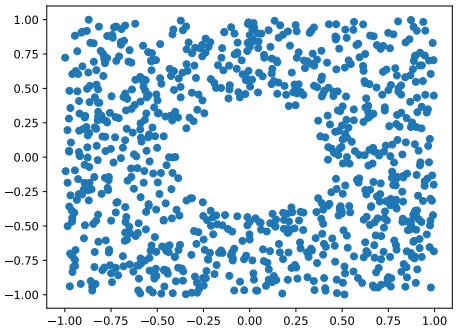


Figure 7.2: Data samples from an annulus.

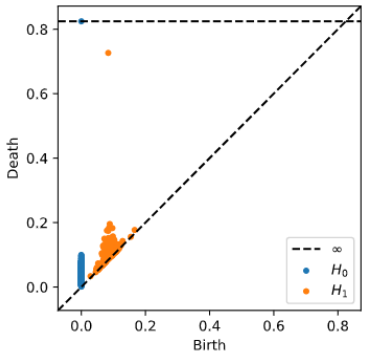


Figure 7.3: The persistence diagram for the sampled data from an annulus. It is clear the topological space of the data has a single component and an 1D hole.

7.3.4 Graph Cut Perspective

The VR complex is essentially an r-neighboring graph if we only consider the vertices and edges. The MoE requires the dataset to be split into finite pieces. This scheme naturally fits into the framework of graph cut problems, which aim to split the graph into several isolated subgraphs by removing some edges. Each subgraph represents a subset of the dataset used to train an expert. Each subset of data also represents a subset of the function domain, denoted as *subdomain* later on. It is important to make sure the function is continuous in each subdomain to avoid the averaging issue from neural networks trained in discontinuous domains, as shown by many examples in Chapter 6.

Noting the each data sample is a pair of input (task parameter of the planning task) and output (optimal trajectories), the argmin discontinuity is approximately represented by pairs of samples that are similar in input but not similar in output. Recalling the argmin maps from parameter space X to trajectory space Z, which are assumed to be equipped with distance function ρ_x and rho_z, respectively, two distance thresholds ϵ and δ are chosen for X and Z. An ϵ-neighboring graph can be built using sampled task parameters, and for each edge in this graph, the weight can be computed as the trajectory distance between its two vertices. The edges with weight greater than δ are labelled as "risky" edges. The goal of graph cut is to make sure in each subgraph no "risky" edges exist by removing edges that connect different subgraphs.

7.3.5 Continuity and Betti Numbers

Considering the motivation is to learn a function from topological space X to Y, we are particularly interested in two topological features: b_0 and b_1 of domain X, image Y, and graph $Z = \{(x, f(x))|x \in X\}$. Here $f(x)$ is the function to be learnt from data. We claim that, by inspecting the Betti numbers b_0, b_1 of X and Z, one can infer if function $f : X \to Y$ is discontinuous. We use notation $b_i(X)$ as the i-th Betti number of space X. We denote $f(X)$ as the image of f on domain X, i.e. $f(X) = \{y|y = f(x), x \in X\}$. The following theorem states that the necessary condition for a function to be continuous is that both b_0 and b_1 of the domain and graph of the function are the same.

Theorem 7.1. A function $f : X \to Y$ where X and Y are topological spaces is continuous only if the domain X and graph $Z = \{(x, f(x))|x \in X\}$ have the same zeroth and first Betti numbers.

A few lemmas are necessary to prove the theorem. The first two are about b_0.

Lemma 7.1. $b_0(X) \leq b_0(Z)$

Proof. For two disjoint components $X_1, X_2 \subset X$, there is no path that connects any $x_1 \in X_1, x_2 \in X_2$. We show there is no path connecting $z_1 \equiv (x_1, f(x_1))$ and $z_2 \equiv (x_2, f(x_2))$ as well by contraction. Assuming $\varphi : [0,1] \to Z$ is a path connecting z_1 and z_2, then the function $\varphi' : [0,1] \to X$ defined as $\varphi'(s) = \varphi(s)_{1:d}$ is also continuous where subscript $1:d$ denotes the first d entry where d is the dimension of X. As a result, φ' is a path in X connecting x_1 and x_2, which contradicts the assumption that x_1 and x_2 are in different components. Since, z_1 and z_2 are in different components as long as x_1 and x_2 are not in the same component, $b_0(X) \leq b_0(Z)$. QED.

Lemma 7.2. If $b_0(X) < b_0(Z)$, f is discontinuous.

Proof. There must be at least one component of X, denoted as $X_1 \subset X$ whose image lies in two components of Z. We assume $x_0, x_1 \in X_1$ while $(x_0, f(x_0))$ and $(x_1, f(x_1))$ are in different components. For path $p(s) : [0,1] \to X_1$ with $p(0) = x_0, p(1) = x_1$, $f(p(s))$ must be discontinuous otherwise $(x_0, f(x_0))$ and $(x_1, f(x_1))$ are in the same component. As a result, f is discontinuous. QED.

These two lemmas show if Z has more connected components than X, f is discontinuous. The following two lemmas are about b_1.

Lemma 7.3. If $b_0(X) = b_1(X), b_0(Z) = 1, b_1(Z) = 0$, f is discontinuous.

Proof. By Lemma 7.1, $b_0(X) = 1$, and, thus, $b_1(X) = 1$ and there is a non-contratible cycle $p(s) : [0,1] \to X$. Consider the mapping $q(s) : [0,1] \to Z$ defined as $q(s) = (p(s), f(p(s)))$. Assuming f is continuous, $q(s)$ is a cycle in Z. Since $b_1(Z) = 0$, $q(s)$ is contratible to a single point. However, using the same way to contract $p(s)$ we may contract $q(s)$ to a single point as well, which contradicts the assumption of non-contratible $p(s)$. As a result, f is discontinuous. QED.

Lemma 7.4. If $b_0(X) = b_1(X), b_1(X) = 0, b_1(Z) = 1$, f is discontinuous.

Proof. Since $b_1(Z) = 1$, we have a non-contratible cycle $q(s) : [0,1] \to Z$, denoted as $(p(s), f(p(s)))$ with $p(0) = p(1)$. As a result, $p(s)$ is a cycle as well which is contratible to a point since $b_1(X) = 0$. Assuming f is continuous, when we contract $p(s)$, $q(s)$ is continuously changing as well so it remains a cycle. When $p(s)$ is contracted into a point, $f(p(s))$ is still a cycle and this cannot happen for continuous f. By contraction, f must be discontinuous. QED.

These two lemmas show if a function's domain and graph have different b_1, it must be discontinuous. Although the proofs are for $b_1 = 1$, they are easily generalizable to other values.

With the four lemmas, it is straightforward to prove Theorem 7.1.

Proof. According to Lemma 7.1, the graph of a function cannot have smaller number of components than its domain. Lemma 7.2 shows f is discontinuous if $b_0(Z) > b_0(X)$. As a result, $b_0(X)$ must equal $b_0(Z)$ for continuous f. Similarly, according to Lemma 7.3 and Lemma 7.4, any change of b_1 between X and Z has to be caused by discontinuous f. As a result, $b_1(X)$ must equal $b_1(Z)$ for continuous f. QED.

Remark 7.1. It is not surprising that a discontinuous function may cause an increase of b_0 and b_1. One example of decreased b_1 is in Fig. 7.4a where X has an 1D hole while Z does not.

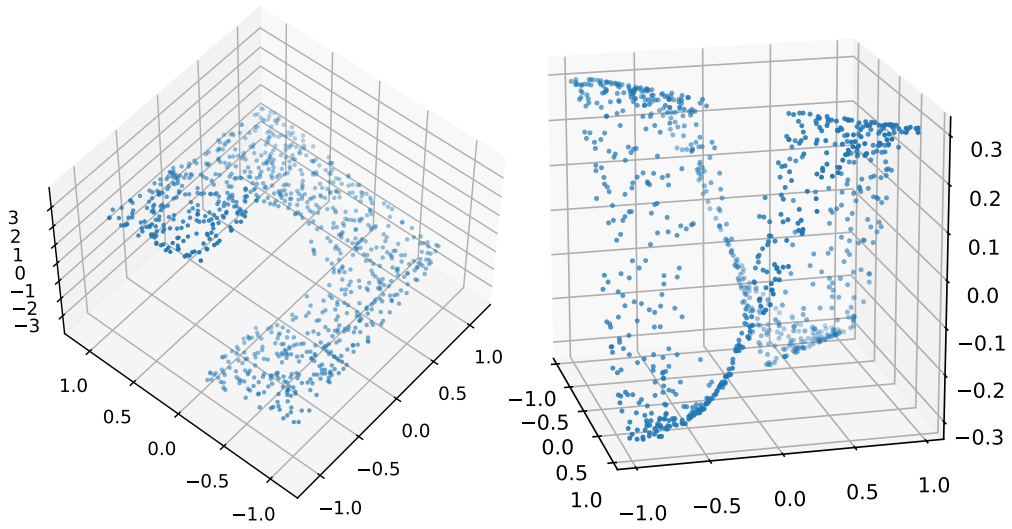
Remark 7.2. One may think the key to discontinuity is the Betti numbers of X and Y. However, the function shown in Fig. 7.4a Fig. 7.4b have the same topology in both X and Y, yet the difference in Z causes a discontinuity seen in Fig. 7.4a.

Remark 7.3. As the proof in Lemma 7.4 shows, one may increase b_1 if the function is continuous in most regions but has a singularity that may map from one point to a cycle. This is indeed the case as shown in Fig. 7.1b. Technically speaking, the domain of the function does not contain the singularity point so $b_1(X)$ is not 0. However, in practice, we are dealing with sampled data and may never sample the singularity point at all. Moreover, empirical results show TDA is not able to capture a 1D hole caused by singularity point so it would still report $b_1(X) = 0$.

Later we show how we can use the information from Betti numbers to analyze the discontinuity of the function and design strategies to better segment them using data.

7.3.6 Discontinuity from Increased b_0

Discontinuity from b_0 occurs when one component is mapped to multiple connected components by a discontinuous function. Such type of function is widely seen in problems with multiple trajectory homotopy classes. For instance, one may avoid an obstacle from both sides, leading to two homotopy classes as shown in Fig. 7.1a. At the location where one has to switch to a different homotopy class, the argmin function has a discontinuity. For instance, in Fig. 7.1a when the start y location moves up, the optimal trajectory switches



(a) $z = \arctan 2(y, x)$ (b) $z = \sin(2 \arctan 2(y, x))$

Figure 7.4: Two function with the same topology in X and Y, but difference in Z causes different behavior in terms of continuity.

to passing the obstacle from another side. This type of discontinuity is well handled using our method. Using the results from TDA, which gives b_0, we can also find a threshold where all major connected components are alive. We can then build the simplicial complex with the selected threshold and perform connected component analysis with the graph (using vertices and edges of the simplicial complex). The dataset is split so that each component is a subgraph representing a subdomain of the function's domain without discontinuity.

7.3.7 Discontinuity from Decreased b_1

In this stage, we assume discontinuity from b_0 has already been handled. An example with decreased b_1 is the 2D obstacle avoidance problem where the start is not restricted to a fixed x, as shown in Fig. 7.5. In this case, the domain of the function has a 1D hole due to the obstacle, but all the trajectories are in the same component without 1D hole structure. This is different from Fig. 7.1a, since path below the obstacle can be continuously transformed into one above the obstacle by moving the start around the obstacle to the left and then up right. Cutting the edges across the horizontal line (right side of the obstacle) is not able to cut the graph into 2 subgraphs since the domain has a hole, i.e. cutting a closed curve once results in an open curve, not two curves. Additional edges have to be cut in order to get isolated subgraphs, which is necessary to guarantee that "risky" edges do not appear in any subgraph. These additional edges can be somewhat arbitrarily chosen but some choices such as edges crossing the horizontal line on the left side of the obstacle may yield better

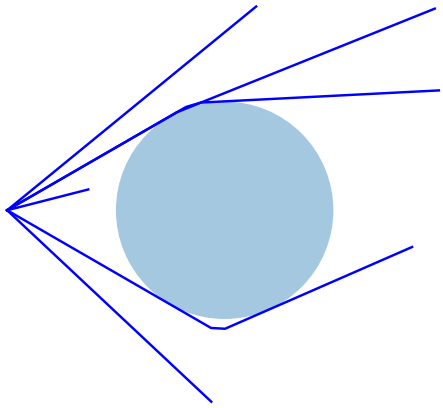


Figure 7.5: 2D obstacle avoidance with free start location. The goal is on the left while the start is free on the right.

conditions for function approximation since the subgraphs are balanced. We propose an iterative algorithm that finds edges that are balanced in terms of distance to the vertices of "risky" edges. The algorithm is shown in Alg. 7.1. It iteratively removes edges that make risky edges exist in a subgraph so that, eventually, all the subgraphs do not contain "risky" edges.

Algorithm 7.1: Split the dataset to remove discontinuity within dataset

Data: Dataset of Samples (x, y), list of risky edges E, r-NN graph
Result: A Split of the dataset with no risky edges

```
Remove risky edges from the graph
while true do
    Perform connected component analysis on the graph
    Find the most risky edge with two vertices being in the same component
    if No edge is found then
        break
    end
    Find the shortest path connecting the two vertices
    Cut the path by removing one edge in the middle
end
Split the dataset according to connected components
```

7.3.8 Discontinuity from Increased b_1

In this section, we focus on the case that f has singularity and causes an increase of b_1. Such singularity is widely seen in 3D obstacle avoidance problems as shown in Fig. 7.1b. In

the neighborhood around a singularity, the function is continuous but has large and rapidly changing gradients, making it hard to capture purely from data. Moreover, this region is small and a uniformly sampled dataset has only a small fraction of data in the singularity region. There is not much that can be done to remedy the large gradients around the singularity, but a good strategy to split the dataset is to split it into a "pizza" shape with the singularity point being the conjunction point for all the pieces. The amount of cuts can be chosen arbitrarily if it is not too small. Note that there is a tradeoff between data size in each piece (presumably, fewer data points make it harder to train good models) and difficulty of function approximation (presuming, again, that a larger piece is harder to fit since it has larger intersection with singularity region, making the dataset more ill-conditioned). In our practice, 8 pieces seem to be a decent choice that balances well between data availability to each expert and difficulty of function approximation.

However, generalizing this idea into an algorithm that does not require expert knowledge on how to perform the "pizza" cuts needs to be addressed. Ideally assuming the increased b_1 is caused by the same geometric singularity, we should be able to automatically find the singularity and cut the data into several pieces in a similar way. Doing so purely from data without expert knowledge is an unsolved problem to the best of the author's knowledge. We propose a method that starts from a non-contractible cycle and computes a generalized angle for each vertex in the cycle. The generalized angle shows the location of a sample within a cycle and samples with similar angles are similar in terms of where they are located in a cycle. However, the generalized angle is defined within $[0, 1]$ to distinguish standard angle within $[0, 2\pi]$ The generalized angle is populated to other vertices using a tree-like structure until every vertex is assigned an angle. Finally we can cut the graph by the angle of each vertex using a uniform grid. This process resembles the "pizza" cut. The details of the algorithm are shown in the following section.

7.4 CYCLE-BASED DATA SPLIT METHOD

Splitting data means grouping data according to similarity or distance. Finding a good distance metric is difficult and often involves considerable trial and error. Classical methods, such as k-Means, assume Euclidean distances and may not work well for higher dimensional trajectory data which lying on some nonlinear manifold. A distance based on topology is desirable since Euclidean distances are only locally useful for data on many nonlinear manifolds. An example is shown in Fig. 7.6 where Euclidean distance assigns a smaller distance to two trajectories from different homotopy classes.

If the function has a singularity near some point, for a point in the neighborhood, the

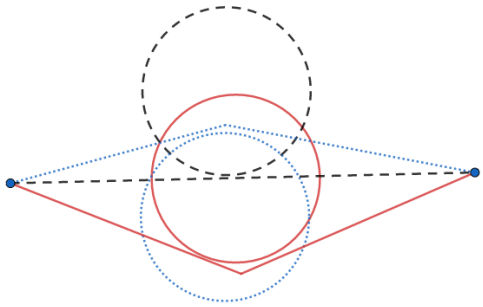


Figure 7.6: 2D obstacle avoidance with a movable obstacle. The black and red trajectory are closer in terms of topology but further in terms of Euclidean distance (compared with black and blue).

direction of perturbation that causes largest change in the objective function's value is from the point to the singularity point. As a result, the data split should try to avoid having edges within the same piece whose distance to the singularity is small. This is exactly why the "pizza" shape piece is a good choice, since it avoids such edges. Admittedly, in each piece, the gradient near singularity is still large so it may be still difficult to precisely capture.

Then the question becomes how to assign an angle-like value to each data. Since a cycle naturally resembles a circle, we can conveniently assign angles to each vertex of the cycle. However, there may be infinite number of cycles that contain one vertex so there may be ambiguity of the angles assigned to each vertex. To resolve such ambiguity, we must select one cycle for each vertex. We select the cycle that has the minimum length and contains the vertex of interest as the representative cycle of that vertex. Moreover, we use the angle of that vertex within its representative cycle as the angle of each vertex. However, to prevent the cycle from shrinking to trivial cases, we require all the cycles be homotopic to each other and not contractible to a single point, i.e. representing the hole. With those rules, the question becomes how to compute the representative cycle for each vertex that is homotopic to existing cycles and how to assign angles to the vertices of the cycle.

To guarantee all representative cycles are homotopic, we propose to use a constructive method, i.e. starting from a cycle (representing the hole) and constructing cycles homotopic to any existing cycle. In this way, it's guaranteed that all cycles are homotopic to each other. The constructive method naturally forms a tree-like structure representing how a cycle is constructed from the first cycle, i.e. the root of the tree. The details of how to construct a homotopic cycle from existing cycles are explained later in Sec. 7.4.2. Supposing all the representative cycles are computed, the tree structure also helps assign angles to each cycle. Several restrictions are used for assigning angles to a cycle:

1. The generalized angles for all vertices are within $[0, 1]$.

2. The angles are monotonically increasing except for the wrap from 1 to 0.

3. For any two consecutive vertices in the cycle, the difference in angles (with proper wrapping) is proportional to the distance between them.

With those rules, it suffices to assign an angle to any vertex of the cycle since all the other angles can be computed accordingly. We first assign angles to the cycle at the root of the tree where some arbitrarily chosen vertex is assigned with angle of 0. Then, following the tree (excluding the root), we assign angles to every node by finding the best alignment with its parent. The best alignment minimizes some distance metric between two cycles. It essentially searches the angle of the representative vertex on a grid that minimizes cycle distance. The details are explained in Sec. 7.4.2. After the tree is traversed (regardless of the order), every data point is assigned an angle which is used for the data split.

7.4.1 Find the Root

First we have to find the root, i.e. a base cycle representing the hole in the structure. Luckily this can be done with the help of TDA algorithms. Ripser [152] computes the birth and death index of topological features and the user has to inspect the persistence diagram in order to determine a lifetime threshold and ignore features with short lifetimes. Assuming the H_1 feature with the longest lifetime is selected, we can directly extract the birth and death index from the return values of Ripser. Moreover, with some modification in the source code, we are able to identify which edge (with weight equal to birth index) closes the cycle so the H_1 feature first appears. We may build a graph that connects every pair of vertices with distance below the birth index of the cycle. Then we perform graph search to find the shortest path between the two vertices of the birth edge that completes the cycle. An example is shown in Fig. 7.7. It's worth mentioning that the cycle tends to be long due to the choice of connection threshold. In fact, with such a threshold, the graph may even have multiple connected components. In order to get a more reasonable cycle (in the notion of length), it is necessary to shrink the cycle to minimum length (while not changing its topology) on a graph with larger connection threshold. However, the connection threshold should not exceed the death index, since it would eliminate the hole otherwise.

We use a simple yet empirically effective method to contract the cycle from TDA into a minimum one. First, with a properly chosen connection threshold, the graph is built. Then we randomly choose two vertices in the cycle and compute the shortest path between them. If the shortest path is different from the path in the cycle, we replace the path in the cycle with the shortest path. In this way, we can reduce the length of the cycle, i.e. contract the

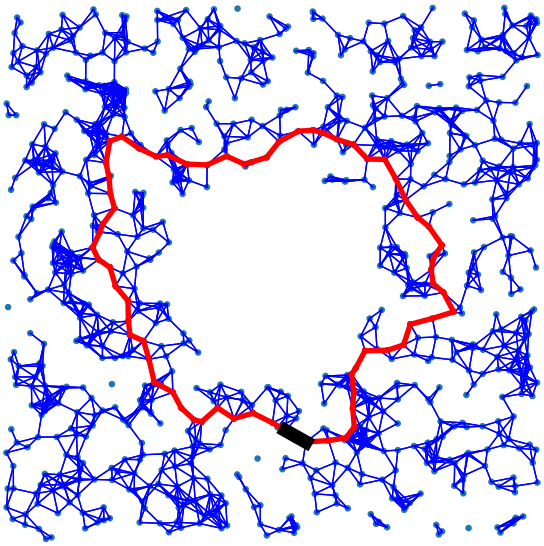


Figure 7.7: The VR complex at the birth of the cycle with longest lifetime and the edge that leads to birth of this cycle (in black).

cycle. However, there are a few practical problems. First, simply replacing a part of the cycle with a path with shorter distance may change the topology and as a result, may eventually shrink the cycle to a point. To prevent such behavior, we only search pairs of neighboring vertices (i.e. at most 25% of the cycle length apart). Second, this shortcut process has to be done repeatedly but the convergence of this method is unknown. Empirical results show this method works reasonably well, as shown in Fig. 7.8. However, more analysis is necessary to establish guarantee of convergence.

7.4.2 Expand Cycle to Neighbors

For a given cycle, it is reasonable to extend the cycle to its neighboring vertices since neighboring vertices are more likely to have similar representative cycles. Here, the neighboring vertices of a cycle are the union of neighboring vertices of each vertex in the cycle. In this section, we introduce how a cycle is propagated to its neighbors.

Homotopic Expansion In order to guarantee homotopy, we use a theorem from [151]. It would be hard to write out the theorem without additional rigorous definitions of complicated concepts. But the main idea is that if an initial cycle is added by the boundary of a simplicial complex composed of triangles, the resulting cycle is homotopic to the initial cycle, as shown in Fig. 7.9 where examples of simplicial complexes composed of one and two triangles are

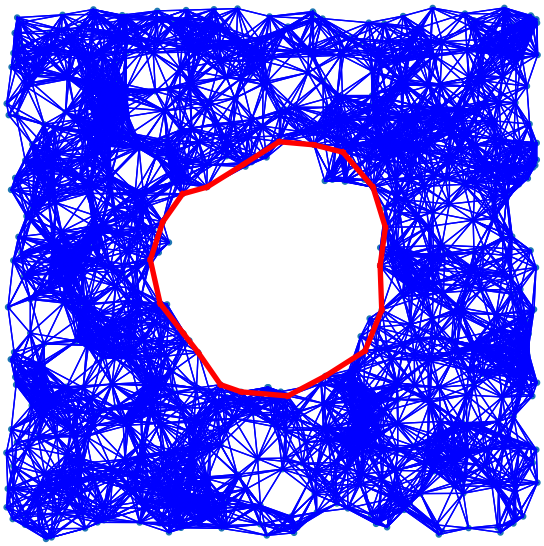


Figure 7.8: The VR complex at the operation threshold (2 times the birth radius) and the shrunk shortest cycle in the graph.

shown. However, it is valid to include any number of triangles as long as they form a simplicial complex. For simplicity, we only consider one-triangle expansion since our cycle optimization algorithm still manages to find the minimum-length cycle , so the multi-triangle expansion are discovered automatically if it leads to a shorter cycle.

Cycle Optimization We propose to use an optimization-based method to find the representative cycle for a vertex from a cycle expanded from a neighboring cycle. The idea is similar to gradient descent and it iteratively finds a homotopic and shorter cycle nearby until improvement halts, i.e. it converges to local minimum.

For a cycle, we find the union of all the neighbors for each vertex of the cycle and construct a subgraph with those vertices. In this way, we can find a region near the cycle and check if a homotopic cycle exists within this region, contains the vertex of interest, and has shorter length. The problem, then, becomes: how to find the shortest cycle with fixed start in a graph that is homotopic to a given cycle. The main difficulty of this task is guaranteeing the homotopy of the result and we propose a dynamic programming based method to handle it.

We denote a cycle with N vertices as $C = (C_0, C_1, \ldots, C_N, C_0)$ and where C_0 is the vertex of interest that has to be within any new cycles. For simplicity, we call it the head of the cycle despite the fact that a cycle has no head. The new cycle, denoted as C', must have $C'_0 = C_0$. To proceed, our DP method tries to find the next vertex for C' within

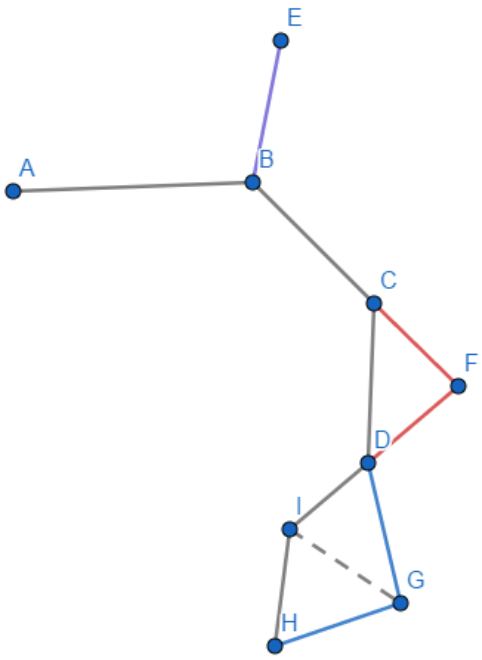


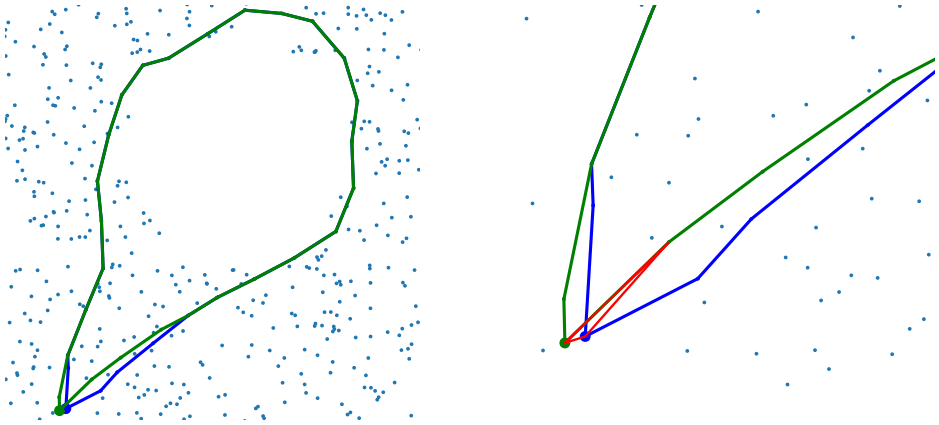
Figure 7.9: Options to expand a cycle to neighboring vertices. The new cycle stays homotopic if CD is replaced with CF and FD or DI and IH are replaced with DG and GH. However, adding BE and EB is excluded.

the neighbors of C_1. To allow for a multi-step path from C_0' to the neighbors of C_1, we further construct a subgraph whose vertices are the union of C_0 and C_1's neighbors. For each neighbor of C_1, a graph search is conducted to compute the shortest path to it from C_0. The shortest path length from C_0 to vertex $v \in \mathcal{N}(C_1)$ is denoted as $w(v)$. Since it's possible that the path may contain more than 2 vertices, the final cycle may have more vertices than the initial cycle. Then, we find the next vertex within $\mathcal{N}(C_2)$ by considering the subgraph obtained by the union of $\mathcal{N}(C_1)$ and $\mathcal{N}(C_2)$. Specifically, for each vertex in $\mathcal{N}(C_1)$, we use Dijkstra's algorithm to find the shortest path to each vertex in $\mathcal{N}(C_2)$, denoted as $d(v_1, v_2)$ with $v_1 \in \mathcal{N}(C_1), v_2 \in \mathcal{N}(C_2)$. The shortest distance to any neighbor of C_2 is thus

$$w(v) = \min_{u \in \mathcal{N}(C_1)} w(u) + d(u, v) \tag{7.4.1}$$

where $w(u)$ has already been computed in the first step. Similarly this can be done for the following vertices of C. Finally the algorithm is again computed back to C_0, i.e. the vertex of interest and it suffices to take $w(C_0)$ as the shortest cycle length. In order to extract the actual cycle, we have to backtrack the search process so some bookkeeping is necessary.

With this method, we can contract the cycle within its neighbors. Similarly to gradient descent, we can iteratively contract the cycle until no improvement can be obtained. This process is guaranteed to converge since the search space is finite (i.e. number of available cycles is finite) and our algorithm is always decreasing the cycle length. However, further analysis is required to establish the rate of convergence and how to avoid local optimum. In practice, the number of iterations is usually small since the new cycle is similar to the previous one as we are considering neighboring vertices of a cycle. An example of cycle expansion from a parent cycle to a neighboring vertex is shown in Fig. 7.10. Notice that, without optimization, the child cycle only replaces one edge of the parent with two edges. Optimization of the child cycle makes it significantly different from the parent cycle.



(a) (b)

Figure 7.10: An example of cycle expansion from a parent cycle (green) to a child (blue). (a) the overall cycle shape. (b) the red triangle details how the expansion is initialized and finally optimized to shorter length.

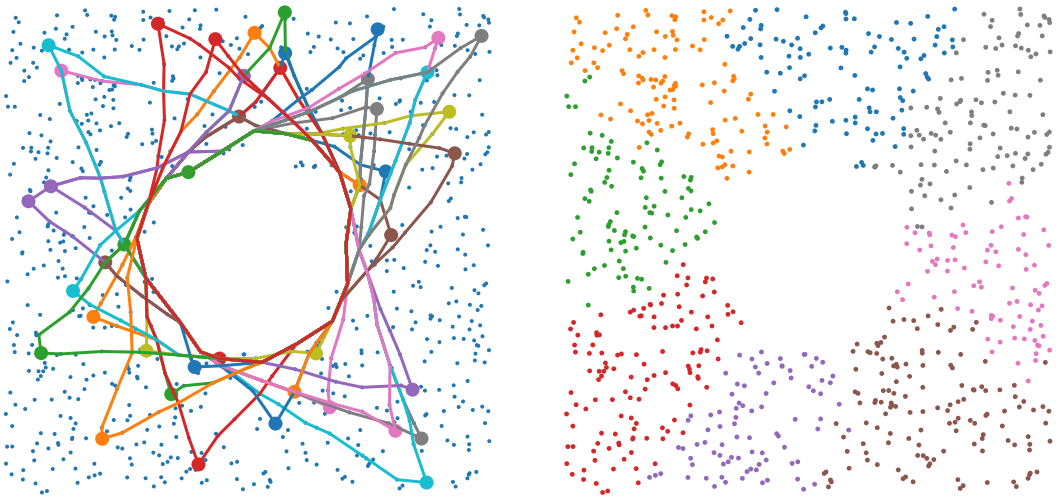
7.4.3 Cycle Alignment

Each cycle is associated with a parent cycle in the cycle expansion algorithm. Assuming the parent cycle is assigned with angles, it's a natural choice to assign angles to the child cycle by some similarity metric of the cycles. Because the rules for assigning angles to a cycle make the cycle angles uniquely determined by its head angle, it suffices to assign an angle to the head that minimizes the distance between two cycles. In order to compute the distance between two cycles, we first define an piecewise linear interpolation function on the parent cycle with its vertex angles as input and data points associated with each vertex as output (with proper wrapping). Then for each vertex of the child cycle, the corresponding data

point in the parent cycle with the same angle is computed by interpolation. The distance from each vertex of the child cycle to the parent cycle is thus computed by the distance to the interpolated data. Then the cycle distance is computed by the sum of the distances from each vertex of the child cycle to the parent cycle. With this distance metric, we can search the best angle for the head of the child cycle on a grid. One reason for grid search is that the distance function is non-convex and non-smooth which hinders the convergence of gradient-based methods. With this subroutine, the angles of all the cycles can be computed by traversing the tree from the root to all the leafs.

7.4.4 Overall Algorithm

With each component ready, we list the overall algorithm in Alg. 7.2 for assigning generalized angles to the data with H_1 topological structure. The key subroutines are listed as well. After computing angles to each data, it's straightforward to split the dataset by selecting a resolution, evenly splitting interval $[0, 1]$, and assigning a group to each data according to the interval wihin which its angle lies. Such a split scheme is similar to the desired "pizza" cut because of the way the generalized angle is computed. Examples of cycles found by the algorithm in a 2D dataset with an 1D hole are shown in Fig. 7.11a. The final data split is shown in Fig. 7.11b. The results show this algorithm is correctly splitting the dataset in a desired way.



(a) Examples of optimized cycles.

(b) Data split using our algorithm where each color denotes a split.

Figure 7.11: Results for the 2D problem with vertical start.

Algorithm 7.2: Compute labels for data with H_1 structure

Data: Samples (x, y), edge threshold ϵ
Result: Labels for each data
Compute the initial cycle as in Alg. 7.3
Contract the initial cycle to get the root cycle C as in Alg. 7.4
Initialize empty priority queue q, empty closed set
for *vertex* C_i *in* C **do**
    Add C_i to closed set
    for *vertex* v *in* $\mathcal{N}(C_i)$ **do**
        if v *not in closed set* **then**
            Add $(d(v, C_i), v, C)$ to q
        end
    end
end
while q *is not empty* **do**
    Pop frontier $(\bar{d}, v, C)$ from q
    if v *not in closed set* **then**
        success, $C' =$ Expand from C to vertex v using Alg. 7.5
        if *success is true* **then**
            Align C' with C and label C'
            Add v to closed set
            for *vertex* $w \in \mathcal{N}(v)$ **do**
                Add $(\bar{d} + d(w, v), w, C')$ to q
            end
        end
    end
end
Return the labels for all vertex in closed set

Algorithm 7.3: Compute the initial cycle

Data: Samples (x, y)
Result: Initial cycle
Concatenate $z = [x, y]$ and perform TDA on z
Find the H_1 feature with longest lifetime, get its birth index α, and birth edge (with vertex u and v)
Construct a r-NN graph with edge threshold α
Remove the birth edge if it is in the r-NN graph
Find shortest path from u to v (e.g. Dijkstra)
Return the cycle by adding edge (u, v) to the shortest path

Algorithm 7.4: Compute the root cycle

Data: Samples z, edge threshold ε, initial cycle C_0, maximum iteration number *maxiter*
Result: Non-contratible root cycle

Construct r-NN graph G with z and edge threshold ε
$i \leftarrow 0, C \leftarrow C_0$
for *i from 1 to maxiter* **do**
    Randomly select a vertex u from C
    Select the vertex v that is int(0.25 $\times$ number of vertices of C) steps away
    Compute the shortest path between u and v on G
    Replace the cycle segment $u \to v$ by the shortest path
end
Return C

Algorithm 7.5: Expand cycle to new vertex

Data: Graph G, parent cycle C, vertex v, maximum iteration maxiter
Result: Representative cycle of v

Compute an initial cycle C (possibly non-shortest) using Alg. 7.6
for *i from 1 to maxiter* **do**
    Compute the shortest homotopic cycle C' around C using Alg. 7.7
    if $C' = C$ **then**
        break
    else
        $C = C'$
    end
end

Algorithm 7.6: Construct homotopic cycle to new vertex

Data: Graph G, parent cycle C, vertex v
Result: A homotopic cycle containing vertex v

Initialize edge list
for *Edge* $e = (a, b)$ *in* C **do**
  if $a, b \in \mathcal{N}(v)$ **then**
    Add $(d(a, v) + d(b, v) - d(a, b), (a, b))$ to edge list
  end
end
if *edge list is empty* **then**
  Return False, None
end
Find the edge in the list whose first entry is smallest
Construct a cycle by replacing edge (a, b) with edges (a, v) and (v, b), denoted as C, make v its head
Return C

Algorithm 7.7: Dynamic programming to find shorter cycle

Data: Graph G, cycle C
Result: A shortest and homotopic cycle around C

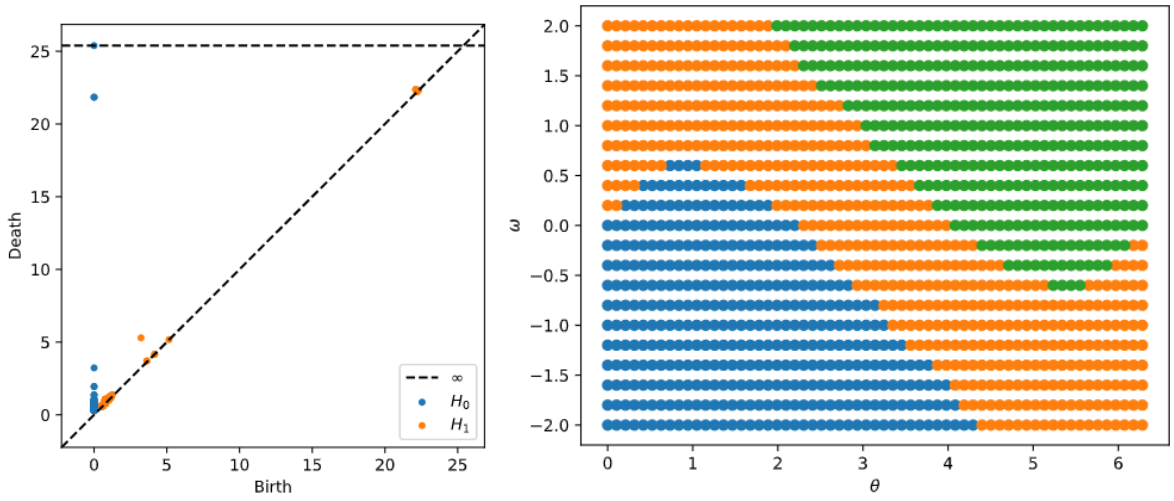
Initialize dictionary w **for** $v \in \mathcal{N}(C_0)$ **do**
  $w(v) = d(v, C_0)$
end
for *Edge* (C_i, C_{i+1}) *in* C **do**
  Construct subgraph using vertices $\mathcal{N}(C_i) \cup \mathcal{N}(C_{i+1})$
  Perform Dijkstra to compute pairwise distance
  for $v \in \mathcal{N}(C_{i+1})$ **do**
    Compute and assign $w(v)$ using Eq. (7.4.1)
  end
end
Get minimum length $w(C_0)$
Backtrack to compute the shortest cycle
Return the shortest cycle

7.5 NUMERICAL EXPERIMENTS

7.5.1 Pendulum Swing-up Revisited

This problem is taken from Chapter 6 and the goal is to show how TDA can be applied to this problem. First we compute the Betti numbers for function domain X and graph Z. It turns out $b_0(X) = 1, b_1(X) = 0, b_0(Z) = 3, b_1(Z) = 0$. The topological structure of X is trivial since we sample the start state uniformly on a grid. The persistence diagram for Z is shown in Fig. 7.12a.

The increase of b_0 shows the existence of a function discontinuity. A threshold when the 3 major components are alive is chosen from the persistence diagram and a graph is built with edge weights below this threshold. The connected component analysis gives 3 components as shown in Fig. 7.12b, the data split from topology agrees with the results in Chapter 6 and the authors' understanding of the problem.



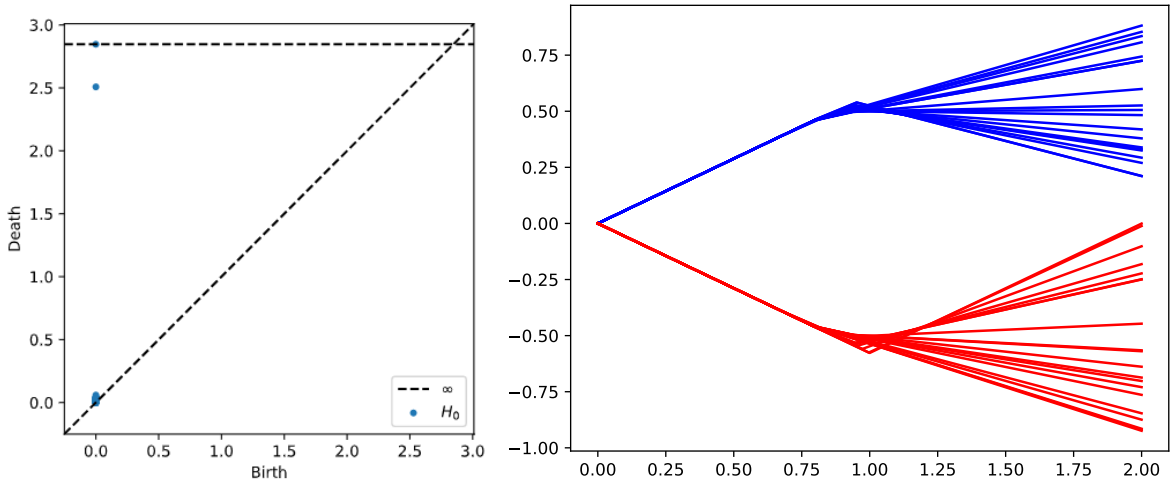
(a) Persistant Diagram (b) Data Split

Figure 7.12: a) Persistence diagram for pendulum swingup problem. 3 connected components can be read from H_0. Note: the 2nd and 3rd component with longest lifetime overlaps. b) Split of the dataset for pendulum swingup problem using our method

7.5.2 2D Obstacle Avoidance with Fixed x

For the problem shown in Fig. 7.1a. We use TDA to get $b_0(X) = 1$ and $b_0(Z) = 2$. As a result, we select the threshold of 1.0 and perform connected component analysis on the simplicial complex. The results are shown Fig. 7.13. Clearly it correctly finds the two

trajectory homotopy classes.



(a) Persistence Diagram (b) Data Split

Figure 7.13: Results for the 2D problem with vertical start.

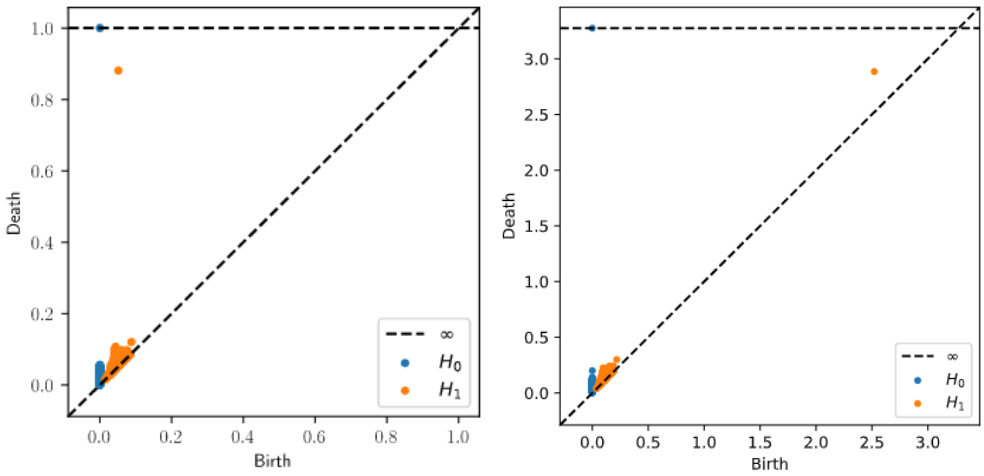
7.5.3 2D Obstacle Avoidance with Free Start

The 2D obstacle avoidance problem with free start location has a different type of topology from the one with fixed x. The TDA result for X and Z are shown in Fig. 7.14, indicating decrease of b_1. The recursive edge cut method is used to handle it.

The results are shown in Fig. 7.15. As the results show, it correctly split the dataset into two pieces to avoid discontinuity. Moreover, the split is balanced to some extent. The results of using this data split to train an MoE and comparing it with SNN, are shown in Tab. 7.1. The results of using k-Means methods in Chapter 6 are similar to this result since this problem is relatively simple.

Table 7.1: Comparison of constraint violation for the 2D obstacle problem with free start.

	SNN	Topo
Avg Vio (m)	0.009	0.001
# Vio	34	8
Max Vio (m)	0.45	0.021



(a) 2D (b) 3D

Figure 7.14: TDA results of domain and graph for the 2D problem with free start. The domain has a single component and a hole while the graph has a single component and no hole.

7.5.4 3D Obstacle Avoidance with Moving Obstacle

To generate the dataset, we randomly sample the obstacle on orthant of a sphere. The obstacle has a fixed radius of 0.3 and is uniformly sampled in that orthant. The start is sampled so that a straight line from the start to the goal has some penetration with the obstacle for the purpose of increasing problem difficulty and sample efficiency with obstacle avoidance information. Compared to a problem with a fixed obstacle such as Fig. 7.1b, this problem is challenging, since the singularity is not a single point but a function of the obstacle location. We generate 80000 data points for training and 4000 for testing. We evaluate the model by computing the constraint violation (i.e. penetration of the trajectory into the obstacle) of the predicted trajectory for the validation set. We use a subset of data to perform TDA to reduce computational cost. Another reason for using a subset is because using only a subset of data may still reveal the topological structure and extract a cycle for us to work with. The results from TDA is in Fig. 7.16, showing the existence of a single 1D hole.

The root cycle is shown in Fig. 7.17 where the colors from blue to red indicate increasing generalized angles. The same with intuition, the cycle is around an obstacle. For this one, the trajectories have similar obstacle locations.

After applying Alg. 7.2 to this problem, a data split is obtained. The MoE can be trained according to this data split and are tested on the validation set. For each validation problem, we use the trained model to predict the path and compute constraint violation as the pene-

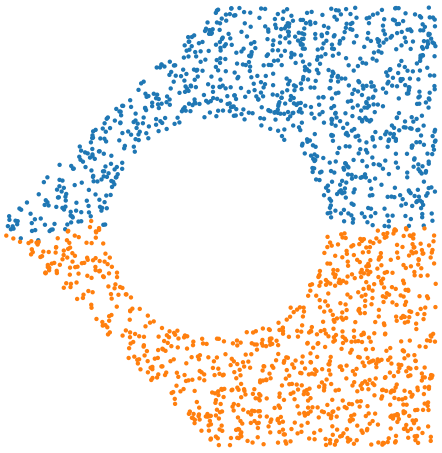


Figure 7.15: Split of data for 2D obstacle problem with free start locations. Each color denotes a subgraph.

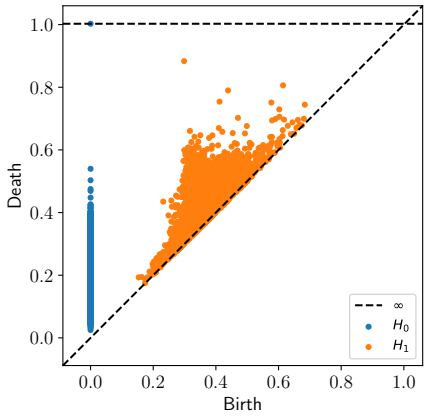


Figure 7.16: Persistence diagram for 3D obstacle problem. It shows $b_0 = b_1 = 1$.

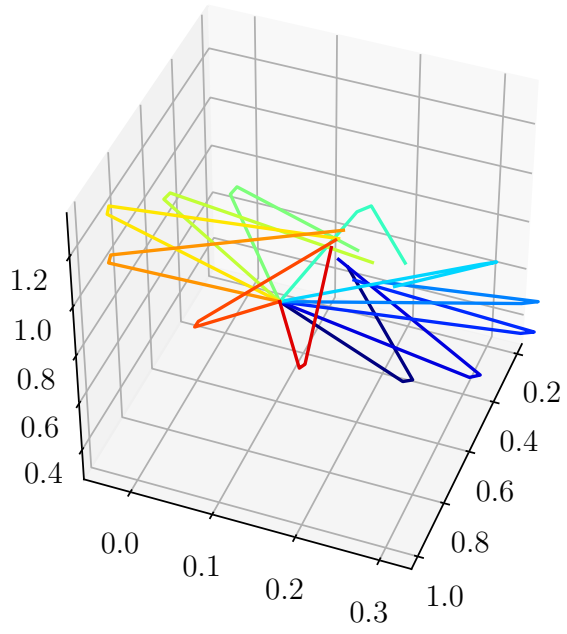


Figure 7.17: The root cycle where colors from blue to red indicate increasing generalized angles.

tration depth of the path into the obstacle. The results are shown in Tab. 7.2. Clearly, our method outperforms all other methods in terms of the mean violation (in all voilated cases). Specifically, our method is much reliable for prediction near singularity regions since it has the smallest maximum violation. The k-Means based data split method outperforms SNN since it is able to separate some amount of discontinuity. However, the maximum violation is similar to SNN and it shows k-Means based method is unable to fully separate discontinuity.

Table 7.2: Constraint violation for different approximation models for 3D obstacle problem. Unit for violation is mm.

	SNN	k-Mean 10	k-Mean 20	k-Mean 40	Cycle
Avg Vio	5.8	4.4	4.0	6.7	3.7
# Vio	1053	530	361	635	477
Max Vio	180.6	193.8	191.2	202.5	49.3

7.5.5 Discussion

We demonstrate the efficacy of our framework in discontinuity detection and data split using examples for each type of discontinuity. They all follow a pipeline of data collection, topological analysis, discontinuity detection, data split, and MoE training. Compared with

SNN, the MoE model is capable of approximating discontinuous functions and our topology-based method correctly splits the data according to estimated discontinuities. This framework improves upon Chapter 6 since there is no requirement of tuning of the number of clusters and has better theoretical foundation.

Problems with increased b_0 are the most straightforward since, for motion planning problems, trajectories from different homotopy classes tend to have large distances so k-Means based method may separate them, even though there is no guarantee about it. However, for problems where intra-distance is larger than inter-distance, k-Means based methods may have worse performance, although this can be partially solved by over-segmentation at the risk of model overfitting in later training stages. On the contrary, topology based methods give the exact data split by discontinuity and is, thus, more convenient. It minimizes the number of connected components so the classifier is easier to train and the region where extrapolation of MoE occurs is minimized.

Problems with decreased b_1 are more complicated and it also bring ambiguity about where to cut non-risky edges. k-Means based methods may still be effective since near discontinuity trajectories have large distances if they should be separated. Again, k has to be fine-tuned to get better results. Our method uses a heuristic method to encourage data balance which needs more theoretical analysis while k-Means method does not directly reason about data balance.

Problems with discontinuities caused by an increase in b_1 are the most difficult, since the singularity is inherently difficult to learn from data. The data split only partially alleviates the problem and relies on extrapolation to handle the large gradient regions. The data split scheme of computing generalized angles helps to realize the desired "pizza" cut and works well in practice. k-Means based method tends to fail at this particular case especially when the singularity point is not constant. In that case, trajectories that are far in terms of generalized angle may be closer (Euclidean distance) than trajectories with similar generalized angles. Our topology-based method can accurately compute trajectory distance based on the 1D hole topology.

Remark 7.4. However, one drawback of using TDA to count connected components is for k components it requires the death indexes of the top k-th H_0 structure to be apparently greater than the rest. Recalling that the death index is the minimum distance between two smaller components, the TDA approach requires that the $(k+1)$-th component merges with other components in early stage. This requires sufficient data density. However, it is difficult to have sufficient data density in high-dimensional spaces and, therefore, the method scales poorly with problem dimensionality. One approach to this problem is to search the death

index in decreasing order and check which threshold gives a data split that minimizes the number of connected components and removes all discontinuity within every component.

The sample size problem occurs to 1D hole topological structures as well. The death index which resembles the "radius" of the cycle has to be much larger than the largest edge length of the cycle when it is formed. Similar to H_0 structure, it requires a sufficient sample density as well. Otherwise the H_1 structure is not recognizable from the persistence diagram.

In sampled data, it is not unusual that data "voids" exist, simply because some regions are not sampled. In that case, TDA may compute some artificial H_1 feature with short lifetimes. However, those voids are detrimental to the cycle contraction method since a large enough search radius is required to jump over them. A larger radius requires more computational resources since in the DP method more neighbors are considered. A poorly chosen radii may result in locally optimal cycles and poor generalized angle computation as well. It is worthwhile studying how dispersion [2] affects the algorithm.

7.6 CONCLUSION

In this chapter we propose a unified framework that combines topological data analysis and MoE for trajectory learning. The discontinuity of the argmin function is detected by computing and comparing the topological features on the function's domain and graph from data. For several types of discontinuity that are widely seen in robotics applications, we study how to identify them from topological features and propose corresponding data split schemes to handle them. For problems with increased number of components, our method is tuning-free and can find the minimum split that removes all discontinuities within each split. For problems with decreased number of 1D hole, our method heuristically finds a relatively balanced split scheme and does not require the tuning of number of clusters like k-Means based approach. For problems with increased number of 1D holes because of singularity, our method computes generalized angles for each data point that approximately represents its location in a cycle. The similarity in generalized angles better preserves similarity in topology which is why it outperforms k-Means based data split method, especially for cases where the singularity point is a function of problem parameters. Future work includes studying problems of higher order holes, complexity and scalability analysis of the methods, and application to more challenging problems.

CHAPTER 8: CONCLUSIONS

In this thesis, the author advocates the exploitation of two structures that exist in a wide range of trajectory optimization problems. With proper methods to exploit them, efficient optimal planners can be built with high reliability.

The convex sub-problem is in general a strong structure, only exists in particular applications and requires domain knowledge to identify it. However, the two types of problems discussed in this thesis are generalizable to lots of applications where polynomial trajectories are used. The bilevel optimization framework is in general difficult to solve but the framework in this thesis behaves surprisingly well because of the differences from general bilevel problems: 1) the lower and upper problem have the same objective function and the smoothness of the objective function is better behaved than argmin, as often seen in other bilevel problems; 2) the upper problem is limited to possess convex constraints only and it simplifies constraint satisfaction by a lot; and 3) the lower problem does not have feasibility issues and does not impose additional limitations to gradient descent of the upper problem. It is unclear and, thus, worthwhile investigating if the convex sub-problem structure widely exists in robotic applications. For future work, one may try to apply this framework to other robotic applications. It is also of great interest to study if the smoothing technique is effective to general bilevel optimization problems where the upper and lower problem have different cost functions. The feasibility issue of the lower problem may be worth investigating as well to see if it imposes challenges to the upper problem and how to handle it. Overall the bilevel framework has great potential since it takes advantage of convex optimization to guarantee feasibility and analytic gradients to speed up the upper optimization. In the worst case scenario, the framework falls back to the approach that heuristically chooses non-convex optimization variables so it cannot be worse than the alternative.

The learning-based framework has a generally wider range of applications since it only requires piecewise continuity of the argmin function which is satisfied in most practical applications. The same with other learning methods, the trajectory learning approach faces the same issues such as data collection, data cleaning, model selection, hyperparameter tuning, and model deployment. In this thesis, we mainly aim to address the issue of the discontinuity of the argmin function. Despite the fact that a function can be discontinuous in arbitrary ways, we have studied several of the most widely seen discontinuity types. Our empirical k-Means approach and theoretical topology-based approach have shown the ability to handle several discontinuity types of interest. The MoE framework has also shown its advantage over discontinuity-agnostic continuous neural network models, especially for

model predictions in regions near discontinuity. Some open questions for trajectory learning include:

- Is there any way to establish a validation error guarantee of the model prediction?;
- Is this approach scalable to problems with high dimensional parameters?;
- Can the requirement of fixed-size input and output dimensionality be lifted?;
- How to quantify trajectory prediction error and establish conditions for guarantees of successful trajectory tracking.

Some open questions for discontinuity in trajectory learning include:

- there are potentially infinite number of discontinuity types, is there a unified framework that handles them all?;
- Is there a training algorithm for MoE such that it automatically detects data discontinuity?; and
- How to handle discontinuities dur to a change in higher order Betti numbers.

Overall, the trajectory learning framework is promising since 1) it moves all computationally expensive nonlinear optimization offline which, in theory, can accept long computation time and low success rates; 2) is flexible enough and does not have strong requirements of problem structures; 3) it predicts approximately optimal trajectory and are easier to be executed in physical experiments and 4) in the worst case scenario, the prediction can be used as an initial guess to optimizers so it cannot make things worse. With more efforts in those open questions, more applications of trajectory learning is foreseeable, with or without MoE.

CHAPTER 9: REFERENCES

[1] H. M. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, S. Thrun, and R. C. Arkin, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[2] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[3] B. Paden, M. Cáp, S. Z. Yong, D. S. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE T. Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016. [Online]. Available: https://doi.org/10.1109/TIV.2016.2578706

[4] A. E. Bryson, "Optimal control-1950 to 1985," *IEEE Control Systems Magazine*, vol. 16, no. 3, pp. 26–33, 1996.

[5] H. J. Sussmann and J. C. Willems, "300 years of optimal control: from the brachystochrone to the maximum principle," *IEEE Control Systems Magazine*, vol. 17, no. 3, pp. 32–44, 1997.

[6] A. E. Bryson, *Applied optimal control: optimization, estimation and control*. CRC Press, 1975.

[7] R. E. Kalman et al., "Contributions to the theory of optimal control," *Bol. soc. mat. mexicana*, vol. 5, no. 2, pp. 102–119, 1960.

[8] D. F. Lawden, *Optimal trajectories for space navigation*. Butterworths, 1963, vol. 3.

[9] A. Bryson Jr and S. E. Ross, "Optimum rocket trajectories with aerodynamic drag," *Journal of Jet Propulsion*, vol. 28, no. 7, pp. 465–469, 1958.

[10] A. E. Bryson and W. F. Denham, "A steepest-ascent method for solving optimum programming problems," *ASME Journal of Applied Mechanics*, vol. 29, 1962.

[11] G. Tang and K. Hauser, "A data-driven indirect method for nonlinear optimal control," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 4854–4861.

[12] F. Jiang, H. Baoyin, and J. Li, "Practical techniques for low-thrust trajectory optimization with homotopic approach," *Journal of guidance, control, and dynamics*, vol. 35, no. 1, pp. 245–258, 2012.

[13] R. Bertrand and R. Epenoy, "New smoothing techniques for solving bang–bang optimal control problems: numerical results and statistical interpretation," *Optimal Control Applications and Methods*, vol. 23, no. 4, pp. 171–197, 2002.

[14] F. Jiang, G. Tang, and J. Li, "Improving low-thrust trajectory optimization by adjoint estimation with shape-based path," *Journal of Guidance, Control, and Dynamics*, pp. 1–8, 2017.

[15] G. Tang and F. Jiang, "Capture of near-Earth objects with low-thrust propulsion and invariant manifolds," *Astrophysics and Space Science*, vol. 361, no. 1, p. 10, 2015.

[16] F. Jiang and G. Tang, "Systematic low-thrust trajectory optimization for a multi-rendezvous mission using adjoint scaling," *Astrophysics and Space Science*, vol. 361, no. 4, p. 117, 2016.

[17] Z. Chi, D. Wu, F. Jiang, and J. Li, "Optimization of variable-specific-impulse gravity-assist trajectories," *Journal of Spacecraft and Rockets*, vol. 57, no. 2, pp. 291–299, 2020.

[18] G. Tang, F. Jiang, and J. Li, "Fuel-optimal low-thrust trajectory optimization using indirect method and successive convex programming," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 4, pp. 2053–2066, 2018.

[19] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[20] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1168–1175.

[21] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems." in *ICINCO (1)*. Citeseer, 2004, pp. 222–229.

[22] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference, 2005.* IEEE, 2005, pp. 300–306.

[23] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[24] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour et al., "Policy gradient methods for reinforcement learning with function approximation." in *NIPs*, vol. 99. Citeseer, 1999, pp. 1057–1063.

[25] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*. Citeseer, 2000, pp. 1008–1014.

[26] Y. Yu, "Towards sample efficient reinforcement learning." in *IJCAI*, 2018, pp. 5739–5743.

[27] Z. Ding and H. Dong, "Challenges of reinforcement learning," in *Deep Reinforcement Learning*. Springer, 2020, pp. 249–272.

[28] A. Ray, J. Achiam, and D. Amodei, "Benchmarking safe exploration in deep reinforcement learning," *arXiv preprint arXiv:1910.01708*, 2019.

[29] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *he 2nd Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, Berkeley, 1951, pp. 481–492.

[30] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. New York: Academic Press, 1981.

[31] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, "User's guide for sol/npsol: A fortran package for nonlinear programming." STANFORD UNIV CA SYSTEMS OPTIMIZATION LAB, Tech. Rep., 1983.

[32] C. R. Hargraves and S. W. Paris, "Direct trajectory optimization using nonlinear programming and collocation," *J. Guidance, Control, and Dynamics*, vol. 10, no. 4, pp. 338–342, 1987.

[33] P. J. Enright and B. A. Conway, "Optimal finite-thrust spacecraft trajectories using collocation and nonlinear programming," *Journal of Guidance, Control, and Dynamics*, vol. 14, no. 5, pp. 981–985, 1991.

[34] D. Tabak and B. C. Kuo, *Optimal control by mathematical programming*. New Jersey: Prentice-Hall, 1971.

[35] D. G. Hull, "Conversion of optimal control problems into parameter optimization problems," *Journal of Guidance, Control, and Dynamics*, vol. 20, no. 1, pp. 57–60, 1997.

[36] P. J. Enright and B. A. Conway, "Discrete approximations to optimal trajectories using direct transcription and nonlinear programming," *Journal of Guidance, Control, and Dynamics*, vol. 15, no. 4, pp. 994–1002, 1992.

[37] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.

[38] A. Heim and O. Von Stryk, "Trajectory optimization of industrial robots with application to computer-aided robotics and robot controllers," *Optimization*, vol. 47, no. 3-4, pp. 407–420, 2000.

[39] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online uav replanning," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2016, pp. 5332–5339.

[40] F. Gao, W. Wu, Y. Lin, and S. Shen, "Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial," in *IEEE International Conference on Robotics and Automation*, 2018, pp. 344–351.

[41] F. Gao, L. Wang, B. Zhou, X. Zhou, J. Pan, and S. Shen, "Teach-repeat-replan: A complete and robust system for aggressive flight in complex environments," *IEEE T. Robotics*, vol. 36, no. 5, pp. 1526–1545, 2020.

[42] G. Tang, W. Sun, and K. Hauser, "Learning trajectories for real-time optimal control of quadrotors," in *Intelligent Robots and Systems (IROS), 2018 IEEE/RSJ International Conference on*. IEEE, 2018, pp. –.

[43] W. Xu, J. Wei, J. M. Dolan, H. Zhao, and H. Zha, "A real-time motion planner with trajectory optimization for autonomous vehicles," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 2061–2067.

[44] X. Hu and J. Sun, "Trajectory optimization of connected and autonomous vehicles at a multilane freeway merging area," *Transportation Research Part C: Emerging Technologies*, vol. 101, pp. 111–125, 2019.

[45] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, 2014.

[46] R. Orsolino, M. Focchi, C. Mastalli, H. Dai, D. G. Caldwell, and C. Semini, "Application of wrench-based feasibility analysis to the online trajectory optimization of legged robots," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3363–3370, 2018.

[47] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1560–1567, 2018.

[48] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers & Chemical Engineering*, vol. 23, no. 4-5, pp. 667–682, 1999.

[49] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[50] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE Intl. Conf. on Robotics and Automation*, 2011, pp. 2520–2525.

[51] J. Tordesillas and J. P. How, "Minvo basis: Finding simplexes with minimum volume enclosing polynomial curves," *arXiv preprint arXiv:2010.10726*, 2020.

[52] B. Açıkmeşe and L. Blackmore, "Lossless convexification of a class of optimal control problems with non-convex control constraints," *Automatica*, vol. 47, no. 2, pp. 341–347, 2011.

[53] D. Verscheure, B. Demeulenaere, J. Swevers, J. D. Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, Oct. 2009.

[54] M. A. Patterson and A. V. Rao, "Gpops-ii: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 1, pp. 1–37, 2014.

[55] T. A. Howell, B. E. Jackson, and Z. Manchester, "Altro: A fast solver for constrained trajectory optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 7674–7679.

[56] G. Still, "Lectures on parametric optimization: An introduction," *Optimization Online*, 2018.

[57] J. Guddat, F. G. Vazquez, and H. T. Jongen, *Parametric optimization: singularities, pathfollowing and jumps*. Springer, 1990.

[58] P. Fernbach, S. Tonneau, and M. Taïx, "Croc: Convex resolution of centroidal dynamics trajectories to provide a feasibility criterion for the multi contact planning problem," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018, pp. 1–9.

[59] K. Hauser, "Fast interpolation and time-optimization with contact," *Int. J. Robotics Research*, vol. 33, no. 9, pp. 1231–1250, 2014. [Online]. Available: https://doi.org/10.1177/0278364914527855

[60] N. R. Kapania, J. Subosits, and J. Christian Gerdes, "A sequential two-step algorithm for fast generation of vehicle racing trajectories," *J. Dynamic Systems, Measurement, and Control*, vol. 138, 04 2016.

[61] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals of operations research*, vol. 153, no. 1, pp. 235–256, 2007.

[62] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. Springer, 2016, pp. 649–666.

[63] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: from classical to evolutionary approaches and applications," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 276–295, 2018.

[64] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[65] A. V. Fiacco, *Introduction to sensitivity and stability analysis in nonlinear programming*. Academic press, 1983.

[66] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.

[67] F. Codevilla, M. Miiller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–9.

[68] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.

[69] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.

[70] S. Schaal, "Learning from demonstration," in *Advances in neural information processing systems*, 1997, pp. 1040–1046.

[71] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2118–2124.

[72] W. Merkt, V. Ivan, T. Dinev, I. Havoutis, and S. Vijayakumar, "Memory clustering using persistent homology for multimodality-and discontinuity-sensitive learning of optimal control warm-starts," *arXiv preprint arXiv:2010.01024*, 2020.

[73] W. Sun, G. Tang, and K. Hauser, "Fast UAV trajectory optimization using bilevel optimization with analytical gradients," *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1811.10753

[74] G. Tang, W. Sun, and K. Hauser, "Enhancing bilevel optimization for uav time-optimal trajectory using a duality gap approach," in *IEEE International Conference on Robotics and Automation*, 2020.

[75] G. Tang and K. Hauser, "Discontinuity-sensitive optimal control learning by mixture of experts," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 7892–7898.

[76] M. Wang, Z. Wang, S. Paudel, and M. Schwager, "Safe distributed lane change maneuvers for multiple autonomous vehicles using buffered input cells," in *IEEE International Conference on Robotics and Automation*, 2018, pp. 1–7.

[77] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo em motion planner," *arXiv:1807.08048*, 2018.

[78] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, pp. 1688–1695, 2017.

[79] Z. Wang, H. Ye, C. Xu, and F. Gao, "Generating large-scale trajectories efficiently using descriptions of polynomials," *arXiv preprint arXiv:2011.02662*, 2020.

[80] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[81] O. Von Stryk, "Numerical solution of optimal control problems by direct collocation," in *Optimal control*. Springer, 1993, pp. 129–143.

[82] Z. Wang, X. Zhou, C. Xu, J. Chu, and F. Gao, "Alternating minimization based trajectory generation for quadrotor aggressive flight," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, July 2020.

[83] J. Tordesillas, B. T. Lopez, and J. P. How, "Faster: Fast and safe trajectory planner for flights in unknown environments," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*. IEEE, 2019, pp. 1934–1940.

[84] X. Xu and P. J. Antsaklis, "Optimal control of switched systems based on parameterization of the switching instants," *IEEE Transactions on Automatic Control*, vol. 49, no. 1, pp. 2–16, 2004.

[85] M. Egerstedt, Y. Wardi, and F. Delmotte, "Optimal control of switching times in switched dynamical systems," in *42nd IEEE International Conference on Decision and Control*, vol. 3, 2003, pp. 2138–2143.

[86] E. R. Johnson and T. D. Murphey, "Second-order switching time optimization for nonlinear time-varying dynamic systems," *IEEE Transactions on Automatic Control*, vol. 56, no. 8, pp. 1953–1957, 2011.

[87] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 93–100.

[88] B. Landry, Z. Manchester, and M. Pavone, "A differentiable augmented lagrangian method for bilevel nonlinear optimization," *CoRR*, 2019. [Online]. Available: https://arxiv.org/abs/1902.03319

[89] H. Pirnay, R. López-Negrete, and L. T. Biegler, "Optimal sensitivity based on ipopt," *Mathematical Programming Computation*, vol. 4, no. 4, pp. 307–331, 2012.

[90] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *Proc. 34th International Conference on Machine Learning - Volume 70*, 2017, pp. 136–145.

[91] S. Gould, B. Fernando, A. Cherian, P. Anderson, R. S. Cruz, and E. Guo, "On differentiating parameterized argmin and argmax problems with application to bi-level optimization," *arXiv:1607.05447*, 2016.

[92] K. Jittorntrum, "Sequential algorithms in nonlinear programming," Ph.D. dissertation, Australian National University, 1978.

[93] S. Boyd and L. Vandenberghe, *Convex optimization*. New York, NY, USA: Cambridge university press, 2004.

[94] D. Davis, D. Drusvyatskiy, S. Kakade, and J. D. Lee, "Stochastic subgradient method converges on tame functions," *Foundations of Computational Mathematics*, pp. 1–36, 2018.

[95] F. H. Clarke, Y. S. Ledyaev, R. J. Stern, and P. R. Wolenski, *Nonsmooth analysis and control theory*. Springer Science & Business Media, 2008, vol. 178.

[96] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5113–5120.

[97] P. E. Gill, W. Murray, M. A. Saunders, and E. Wong, "User's guide for SNOPT 7.7: Software for large-scale nonlinear programming," Department of Mathematics, University of California, San Diego, La Jolla, CA, Center for Computational Mathematics Report CCoM 18-1, 2018.

[98] P. E. Gill, W. Murray, M. A. Saunders, and E. Wong, "User's guide for SQOPT 7.7: Software for large-scale linear and quadratic programming," Department of Mathematics, University of California, San Diego, La Jolla, CA, Center for Computational Mathematics Report CCoM 18-2, 2018.

[99] W. Sun, G. Tang, and K. Hauser, "Fast uav trajectory optimization using bilevel optimization with analytical gradients," in *2020 American Control Conference*. IEEE, 2020, pp. 1–6.

[100] Q. Pham, "A general, fast, and robust implementation of the time-optimal path parameterization algorithm," *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1533–1540, Dec. 2014.

[101] G. Tang, W. Sun, and K. Hauser, "Time-optimal trajectory generation for dynamic vehicles: A bilevel optimization approach," in *Intelligent Robots and Systems (IROS), 2019 IEEE/RSJ International Conference on*. IEEE, 2019, pp. 0–7.

[102] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, Mar. 2006. [Online]. Available: https://doi.org/10.1007/s10107-004-0559-y

[103] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM J. on Optimization*, vol. 12, no. 4, pp. 979–1006, Apr. 2002. [Online]. Available: http://dx.doi.org/10.1137/S1052623499350013

[104] J. h. Jeon, R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma, "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving," in *2013 American Control Conference*, June 2013, pp. 188–193.

[105] F. Bayer and J. Hauser, "Trajectory optimization for vehicles in a constrained environment," in *IEEE Conf. Decision and Control (CDC)*, Dec. 2012, pp. 5625–5630.

[106] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *PACIFIC J. MATHEMATICS*, 1990.

[107] W. Xu, J. Wei, J. M. Dolan, H. Zhao, and H. Zha, "A real-time motion planner with trajectory optimization for autonomous vehicles," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 2061–2067.

[108] T. Lipp and S. Boyd, "Minimum-time speed optimisation over a fixed path," *International Journal of Control*, vol. 87, no. 6, pp. 1297–1311, 2014. [Online]. Available: https://doi.org/10.1080/00207179.2013.875224

[109] E. Velenis and P. Tsiotras, "Optimal velocity profile generation for given acceleration limits; the half-car model case," in *IEEE Int. Symp. Industrial Electronics, 2005. ISIE 2005.*, vol. 1, 2005, pp. 361–366.

[110] J. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *Int. J. Robotics Research*, vol. 4, no. 3, pp. 3–17, 1985. [Online]. Available: https://doi.org/10.1177/027836498500400301

[111] H. Pham and Q. Pham, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 645–659, June 2018.

[112] L. Vandenberghe, "The cvxopt linear and quadratic cone program solvers," 2010. [Online]. Available: http://www.seas.ucla.edu/~vandenbe/publications/coneprog.pdf

[113] N. Jetchev and M. Toussaint, "Fast motion planning from experience: trajectory prediction for speeding up movement generation," *Autonomous Robots*, vol. 34, no. 1-2, pp. 111–127, Jan. 2013.

[114] K. Hauser, "Learning the problem-optimum map: Analysis and application to global optimization in robotics," *IEEE Trans. Robotics*, vol. 33, no. 1, pp. 141–152, Feb. 2017.

[115] R. P. Russell, "Primer vector theory applied to global low-thrust trade studies," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 2, pp. 460–472, 2007.

[116] A. Cassioli, D. Di Lorenzo, M. Locatelli, F. Schoen, and M. Sciandrone, "Machine learning for global optimization," *Computational Optimization and Applications*, vol. 51, no. 1, pp. 279–303, 2012.

[117] J. Pan, Z. Chen, and P. Abbeel, "Predicting initialization effectiveness for trajectory optimization," in *IEEE Intl. Conf. Robotics and Automation*, 2014.

[118] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis: a survey," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289–309, 2014.

[119] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters, "Trajectory planning for optimal robot catching in real-time," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 3719–3726.

[120] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos, "The explicit solution of model predictive control via multiparametric quadratic programming," in *Proc. American Control Conf.*, vol. 1–6, 2000, pp. 872 – 876.

[121] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, "User guide for minpack-1," CM-P00068642, Tech. Rep., 1980.

[122] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," in *Pattern Analysis and Machine Intelligence*, vol. 36, 2014.

[123] H. Maurer and D. Augustin, "Sensitivity Analysis and Real-Time Control of Parametric Optimal Control Problems Using Boundary Value Methods," in *Online Optimization of Large Scale Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 17–55.

[124] Z. Xie, C. K. Liu, and K. K. Hauser, "Differential dynamic programming with nonlinear constraints," in *IEEE Int'l. Conf. Robotics and Automation*, Sep. 2016, pp. 1–8.

[125] R. Ritz, M. Hehn, S. Lupashin, and R. D'Andrea, "Quadrocopter performance benchmarking using optimal control," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 5179–5186.

[126] T. Tomić, M. Maier, and S. Haddadin, "Learning quadrotor maneuvers from optimal control and generalizing in real-time," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1747–1754.

[127] P. Foehn, D. Falanga, N. Kuppuswamy, R. Tedrake, and D. Scaramuzza, "Fast trajectory optimization for agile quadrotor maneuvers with a cable-suspended payload," in *Robotics: Science and Systems*, 2017, pp. 1–10.

[128] M. Geisert and N. Mansard, "Trajectory generation for quadrotor based systems using numerical optimal control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 2958–2964.

[129] J. Förster, M. Hamer, and R. D'Andrea, "System identification of the crazyflie 2.0 nano quadrocopter," B.S. thesis, ETH Zurich, 2015.

[130] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "Osqp: An operator splitting solver for quadratic programs," in *UKACC 12th International Conference on Control (CONTROL)*, 2018, pp. 339–339.

[131] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[132] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[133] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.

[134] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[135] R. Murray-Smith and T. Johansen, *Multiple model approaches to nonlinear modelling and control*. CRC press, 1997.

[136] R. R. Selmic and F. L. Lewis, "Neural-network approximation of piecewise continuous functions: application to friction compensation," *IEEE transactions on neural networks*, vol. 13, no. 3, pp. 745–751, 2002.

[137] B. Llanas, S. Lantarón, and F. J. Sáinz, "Constructive approximation of discontinuous functions by neural networks," *Neural Processing Letters*, vol. 27, no. 3, pp. 209–226, 2008.

[138] B. Tang, M. I. Heywood, and M. Shepherd, "Input partitioning to mixture of experts," in *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, vol. 1. IEEE, 2002, pp. 227–232.

[139] H. Cui, V. Radosavljevic, F.-C. Chou, T.-H. Lin, T. Nguyen, T.-K. Huang, J. Schneider, and N. Djuric, "Multimodal trajectory predictions for autonomous driving using deep convolutional networks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2090–2096.

[140] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 1177–1178.

[141] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, Apr. 2012.

[142] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[143] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[144] L. McInnes and J. Healy, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[145] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. PéRez, and I. Perona, "An extensive comparative study of cluster validity indices," *Pattern Recognition*, vol. 46, no. 1, pp. 243–256, 2013.

[146] H. Zhang, J. E. Fritts, and S. A. Goldman, "Image segmentation evaluation: A survey of unsupervised methods," *computer vision and image understanding*, vol. 110, no. 2, pp. 260–280, 2008.

[147] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg et al., "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[148] S. Bhattacharya, M. Likhachev, and V. Kumar, "Topological constraints in search-based robot path planning," *Autonomous Robots*, vol. 33, no. 3, pp. 273–290, 2012.

[149] S. Bhattacharya, R. Ghrist, and V. Kumar, "Persistent homology for path planning in uncertain environments," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 578–590, 2015.

[150] M. Farber, *Invitation to topological robotics*. European Mathematical Society, 2008, vol. 8.

[151] H. Edelsbrunner and J. Harer, *Computational topology: an introduction*. American Mathematical Soc., 2010.

[152] U. Bauer, "Ripser: efficient computation of vietoris-rips persistence barcodes," *arXiv preprint arXiv:1908.02518*, 2019.