

Nonconvex Trajectory Optimization using Trajectory Sensitivities: Application to Personalized Autonomous Driving

by

Xiaofei Wu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2025

© Xiaofei Wu 2025

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contribution

Xiaofei Wu was the sole author of this thesis, which was written under the supervision of Professor Michael Fisher and Professor Stephen L. Smith. This thesis includes first-authored peer-reviewed material that will appear in conference proceedings of 2025 European Control Conference (ECC).

Abstract

Autonomous Driving (AD) has been studied in the past decade and has been gradually deployed in everyday life. A key factor in increasing people’s level of acceptance is trust, which may be enhanced by personalized autonomous driving. One way to design personalized autonomous vehicles is by mimicking the driver’s own driving style while driving safely. Many existing works explore learning-based approaches to achieve this goal. However, the performance of these methods is highly dependent on sample efficiency, and it is usually difficult to enforce safety guarantees. To mitigate these difficulties, this thesis proposes an autonomous vehicle control framework in the form of a parameterized nonconvex trajectory optimization problem with a bilevel structure, where the upper-level models the driving style of a target driver and the lower-level performs vehicle motion planning. Therefore, the focus of this work is the formulation of this parameterized nonconvex trajectory optimization problem and its solution methods, discussed under an application scenario of personalized autonomous vehicles.

The lower-level of the bilevel programming problem solves a trajectory optimization problem. The nonlinear dynamic of the vehicle model leads to challenging nonconvex trajectory optimization problems. Many existing approaches formulate them as multistage programs and rely on derivatives of each stage to obtain a local approximation at each iteration, in which case the quality of approximation when solving the optimization program has significant impact on convergence behavior.

In this work, we develop a novel approach for obtaining improved local approximations when solving nonconvex trajectory optimization problems. By performing an input-to-state reformulation of system dynamics, we use trajectory sensitivities, which are derivatives of the entire system trajectory with respect to control inputs, to form local approximations. This novel approximation method, when used to solve optimization problem and to linearize the constraints, results in less approximation error than the traditional approach, while the latter has accumulating numerical errors for multi-stage planning problems. Local convergence guarantees for the proposed method are presented for nonconvex optimization problems with input-affine inequality constraints. The method is applied to generate trajectories for an autonomous vehicle that are not dynamically feasible and is extended to include a scenario with static obstacles that introduces nonconvex constraints.

The upper-level of the bilevel programming problem models the driving style of the target driver by minimizing the difference between human driving data and motion planning results. The decision variables are weight factors that characterize the driving style and are used to parameterize the lower-level objective, hence affecting its planning results.

We adopt a gradient-based approach to solve this problem. However, differentiability is not guaranteed given the bilevel structure and the nonconvex lower-level solution mapping, so we use subgradient "descent" to generalize gradient descent for non-differentiable functions. The quotation marks suggest the fact that subgradient methods are not necessarily monotone. Therefore, a projected subgradient update algorithm is adopted to solve the upper-level problem.

When learning-based approaches may fail in rare or unseen scenarios, our proposed method with an embedded vehicle model will continue to work. In addition, the optimization framework with dynamical and safety constraints ensures driving safety. The lower-level motion planner has been simulated on a variety of reference paths and compared with the traditional sequential quadratic programming with conventional first-order Taylor approximation to outperform in approximation accuracy, allowable trust-region radius, iterations to converge, and total solver time. Furthermore, our method is less prone to failure when handling multiple obstacles with a complex reference. The upper-level problem is also simulated to solve tracking problems and obstacle avoidance problems, demonstrating its ability to mimic the driving style of a target driver.

Acknowledgements

I would like to take this opportunity to acknowledge all those who gave me support and made this work possible.

I greatly appreciate my supervisors, Dr. Michael Fisher and Dr. Stephen L. Smith, for the opportunity to join their research groups and work with them. Their invaluable guidance and insightful feedback have pointed me to new directions and are instrumental in shaping this research work. They have provided me with many constructive suggestions when I define my research problem, explore solution methods, and choose viable directions. Their expertise and timely support have given me confidence to proceed when faced with challenges. They helped me identify the mistakes in the theoretical analysis and limitations in the experiment design to better refine my approach. Most importantly, their enthusiasm and dedication to academic research have greatly inspired me. I gain an increasing level of accomplishment from the process of doing research, not just from the outcome. I am very excited to continue my Ph.D. study with them.

I also want to thank my colleagues in the DOCS group and the Autonomous Systems Lab. Their positive work attitudes and cooperative skills are what I have learned a lot from, and I appreciate all the in-depth discussions we have shared and the mutual support we give each other. I want to thank Elin, Garrett for decorating the office, organizing events, and making the office feel like home. To Penny, thank you for sharing your knowledge on classical control theory and for offering help. To Peng and Pamul, thank you for bringing new vibes to our groups. To Sam and Zhong from a math background, your unique ideas and insights are greatly appreciated. In addition, I want to thank Barry, Jack, Megnath, Ruiqi and Shamak for sharing valuable research experiences. I also want to thank Avraiem, Anna, Ninghan and Steven for the fun memories of taking the same course and "debating" about assignments. To Krishna and Praneeth, your research work have been an inspiration to me.

I would like to extend my gratitude to my readers, Dr. Roberto Guglielmi and Dr. Yash Vardhan Pant as well. Thank you for reading my thesis, attending my seminar, and sharing insightful and constructive feedback. Your comments on problem formulation, simulation design, and future research directions are very inspiring and help me to better refine this thesis.

Finally, I want to thank my parents for their love and company. During my academic journey, they have supported all my decisions and encouraged me to pursue my goals. I also want to thank my grandparents, who have always been proud of me for any tiny achievement. I am incredibly grateful to be born in this loving family.

Dedication

I would like to dedicate this thesis to my loving family. The encouragement and support of my parents and grandparents has made it possible.

Table of Contents

Author's Declaration	ii
Statement of Contribution	iii
Abstract	iv
Acknowledgements	vi
Dedication	vii
List of Figures	xii
List of Tables	xiv
List of Abbreviations	xv
1 Introduction	1
1.1 Background	1
1.2 Literature Review I	2
1.2.1 Human Driver Mimicking for Better AD	3
1.2.2 Human Driver Mimicking for Personalized AV Control	4
1.2.3 Human Driver Mimicking: A Brief Summary	4
1.3 Literature Review II	5

1.4	Contributions	7
1.5	Organization of the Thesis	9
2	Preliminaries	10
2.1	Problem Statement	10
2.2	Problem Formulation	10
2.2.1	Vehicle Motion Planning Problem	11
2.2.2	Driving Style Imitating Problem	14
2.3	Key Results from Related Work	16
2.3.1	Sequential Quadratic Programming	16
2.3.2	Tensor Operation, Partial Derivatives, and Einstein Summation Notation	17
2.3.3	Trajectory Sensitivity	18
2.3.4	Subgradient and Subdifferentials	18
2.3.5	Additional Notation	19
3	Motion Planning Problem	20
3.1	Linearization of System Dynamics	20
3.1.1	Input-to-state map	20
3.1.2	Trajectory Sensitivities Computation	21
3.1.3	TS-based linearization	23
3.1.4	Comparison of Approximation Results	24
3.2	Modified SQP Method	26
3.2.1	Algorithm Design	26
3.2.2	Comparison of Approximation Error	27
3.3	Convergence Analysis	29
3.4	Adaptation to a Tracking Controller	36

4	Driving Style Imitating Problem	38
4.1	Sensitivity Analysis	39
4.1.1	Derivation of Expressions	39
4.1.2	Second-Order Trajectory Sensitivities Computation	41
4.2	Upper-Level Subgradient Computation	42
4.3	Algorithm Design	42
5	Simulation	45
5.1	Motion Planning Problem: Setup	45
5.1.1	Single-Track Bicycle Model	45
5.1.2	Initialization	47
5.1.3	Extension to Collision Avoidance	47
5.2	Motion Planning Problem: Simulation	48
5.2.1	Convergence behaviour and Trust Region Radius	49
5.2.2	Convergence behaviour and Initial Inputs	50
5.2.3	Example with Collision Avoidance	51
5.3	Driving Style Imitating Problem: Setup	54
5.3.1	Dynamic bicycle Model	54
5.3.2	Objective and Subgradient	55
5.3.3	Initialization and Parameter Selection	56
5.4	Driving Style Imitating Problem: Simulation	58
5.4.1	Reference Tracking	59
5.4.2	Obstacle Avoidance	62
6	Conclusion	69
	References	72
	APPENDICES	79

A	PDF Plots From Matlab	80
A.1	Analysis of LL Subgradient Computation	80
A.2	LL Subgradient without Input-to-State Map	81
B	PDF Plots From Matlab	84
B.1	Vehicle Parameters	84
B.2	Sampled Weights and Obstacle Positions	85
B.3	Additional Figures	86

List of Figures

5.1	Total iterations to converge. LSQP is shown for a trust region of 0.3, while Sequential Quadratic Programming with Trajectory Sensitivity (SQP-TS) is shown for a trust region of both 0.3 and 0.5.	50
5.2	Total solver time (blue and black lines) and iterations to converge (green and red lines) for different constant initial inputs using LSQP (solid lines) and SQP-TS (dotted lines) methods.	51
5.3	Straight reference path with an obstacle in which both the proposed method and LSQP converge.	52
5.4	Dubins reference path with an obstacle in which both the proposed method and LSQP converge.	53
5.5	Complex path with multiple obstacles in which SQP-TS converges but LSQP fails.	53
5.6	Personalized vehicle motion planning following a curved reference path using one demonstration.	60
5.7	Personalized vehicle motion planning following a curved reference path using 10 demonstrations.	61
5.8	Personalized vehicle motion planning following a complex reference path using one demonstration.	62
5.9	Personalized vehicle motion planning following a complex reference path using the first 5 of 10 demonstrations.	63
5.10	Personalized vehicle motion planning following a complex reference path using the last 5 of 10 demonstrations.	64
5.11	Personalized vehicle motion planning following a curved reference path using one demonstration.	65

5.12	Personalized vehicle motion planning following a curved reference path using one demonstration.	65
5.13	Personalized vehicle motion planning at a roundabout with obstacle using Demonstrations 1-4.	66
5.14	Personalized vehicle motion planning at a roundabout with obstacle using the first Demonstrations 5-8.	67
5.15	Personalized vehicle motion planning at a roundabout with obstacle using the first Demonstrations 9-10.	68
B.1	Reference paths and simulated driving data following a curved path using 10 demonstrations.	86
B.2	Reference paths and simulated driving data following a complex path using 10 demonstrations.	87
B.3	Reference paths and simulated driving data at a roundabout with obstacle using 10 demonstrations.	87

List of Tables

5.1	Parameters of projected subgradient update algorithm	57
B.1	Definition of Vehicle Parameters	84
B.2	Sampled weights for tracking of a curved path	85
B.3	Sampled weights for tracking of a complex path	85
B.4	Sampled weights and obstacle positions for tracking of a complex path . .	86

List of Abbreviations

AI Artificial Intelligence 4, 6, 68

AV Autonomous Vehicle 1–11, 36, 52, 67, 68

BP Bilevel Programming 15, 18, 36, 40, 43, 52, 58, 60, 63–65, 67

CT Continuous-Time 44, 52, 53

DT Discrete-Time 6, 11, 12, 18, 53

KKT Karush–Kuhn–Tucker 37

LL Lower-level 7, 8, 18, 20, 36–38, 40–42, 52, 54–56, 58, 61, 68

NLP Nonlinear Programming 6, 12, 21, 29

ODE Ordinary Differential Equation 11, 18, 52

QP Quadratic Programming 16, 27, 33

SQP Sequential Quadratic Programming 6–8, 16, 17, 20, 21, 23, 26, 43, 45, 46, 67

SQP-TS Sequential Quadratic Programming with Trajectory Sensitivity xii, 26, 27, 29, 43, 46–50, 52, 54, 56, 67

TS Trajectory Sensitivities 20, 24–26, 28, 37–40, 42, 54

UL Upper-level 7, 8, 18, 36, 37, 40–42, 52–55, 63, 68, 78

Chapter 1

Introduction

1.1 Background

The past few years has seen significant development in autonomous driving. [Autonomous Vehicle \(AV\)](#) are now becoming safer, more intelligent, and more accessible [60]. However, despite the technological breakthrough, autonomous driving still faces some resistance from the public. In many cases, people’s preferences for autonomous driving are hampered by mistrust or unfamiliarity [35]. To alleviate these negative feelings towards AV, researchers study the relationship between people’s acceptance of AV and the way AV operates. They find that people’s expectations of an AV are likely affected by their own driving style [24].

The driving style of a driver is the collection of characteristics that influence their decision making in some specific driving scenarios [13]. An example of a driving style is cautious driving, where a driver tends to slow down at traffic intersections and always maintain a sufficient amount of headway distance. Driving styles can be classified by the level of aggression as aggressive, cautious, and normal driving, or based on the driver’s emotional tendencies as risky, anxious, dissociative, and distress reduction driving [36] [54]. Existing work shows that an AV that mimics the user’s own driving style likely increases the level of trust and comfort [52]. When traversing the same path, drivers with different driving styles produce different vehicle trajectories. Therefore, our goal is to design a personalized AV controller that improves the user’s ride experience by producing vehicle trajectories that match the user’s own driving style.

Inspired by the need of a personalized AV controller, we treat the problem as a parameterized nonconvex trajectory optimization problem, whose formulation and solution

method are the focus of this thesis. An important component of a personalized AV controller, the vehicle motion planner, is considered a trajectory optimization problem. However, to achieve accurate predictions of vehicle states for more reliable vehicle control, a nonlinear vehicle model with complex dynamics must be used. In addition, other input and state constraints, such as obstacle avoidance constraints, may also be nonlinear. This nonlinearity results in a challenging nonconvex optimization problem. Thus, our other goal is to develop an algorithm that solves nonconvex trajectory optimization problem and produces dynamically feasible and obstacle-free vehicle trajectories.

In line with our two main goals, the subsequent literature review section is divided into two parts. In this first part, we review previous work related to personalized AV control. This includes works with different interpretations of personalization, while focusing on works that achieve personalized AV control by mimicking the driving styles of human drivers. Various approaches to achieve driving style imitation are discussed, as well as their strengths and limitations. In the second part, we review existing work on vehicle motion planning. We mention a variety of methods that address this problem, but pay special attention to the trajectory optimization formulation along with its corresponding solution methods.

1.2 Literature Review I

Although there is much research work on the topic of autonomous driving, the study of personalized AV control does not receive equal attention. To the best of our knowledge, there has not been a formal definition of what personalized AV control should do, and this idea is approached differently. The work of J. Ling *et al.* adopts an Emotion Preference Style Adaptation (EPSA) framework, which achieves control personalization based on the emotional feedback of users [35]. This method is able to actively adapt the AV driving behaviour to better fit the needs of the users. However, it assumes a relationship between the user’s emotion and preferred driving style and does not consider the user’s own driving style.

Other researchers make use of the categorization of driving styles, especially the level of aggression, which has been shown to have a significant impact on people’s preference for AV [3] [61]. The work of J. Eriksson and L. Svensson adopt a linear quadratic controller, whose parameters can be tuned according to the driver’s preferred level of aggression and therefore improves ride comfort [53]. However, this approach requires users to have knowledge of various levels of aggression and may even require some test rides for the controller to work well. More importantly, categorization-based methods can be very limiting depending

on the definition of each category, and it is hard to design a categorization scheme that considers all possible combinations of driving habits.

In addition to the two methods discussed above, a considerable portion of the existing work is related to personalized AV controller that favors the driver’s driving style and attempts to imitate it [17, 31, 5]. Imitation-based methods are capable of generating personalized AV commands without deliberate user input. Although personalized AV control is a relatively new research topic, the study of human driver mimicking is of wide interest at an earlier stage for a different purpose. Many prior works learn from human drivers to improve the autonomous driving technology, so that AVs operate in a safer, more intelligent and more human-like way [19, 21, 59]. In this section, we review work on human driver imitation, both for the purpose of better autonomous driving and personalized AV control.

1.2.1 Human Driver Mimicking for Better AD

We first discuss research on human driver mimicking for improved autonomous driving. In this type of study, generic and abstract human driving behaviours are extracted from a large dataset, studied, and mimicked [42]. One common approach to achieve this is Reinforcement Learning (RL) and its extensions. For example, Z. Wu *et al.* adopt Inverse Reinforcement Learning (IRL) to extract a reward function from human driving data (i.e. expert behaviours), using which a policy for autonomous driving may be derived [58]. However, reinforcement-learning-based approaches generally have high computation costs, and the learned reward function using inverse reinforcement learning may not be unique given the same expert demonstration [42].

The method of Behaviour Cloning (BC) is also used to achieve human driver imitation. In behaviour cloning, a model policy is learned using supervised training to match experts’ actions with system states [42] [46]. An example is the work of S. Sharma *et al.*, where a deep convolutional network is trained based on observations of human driving [50]. These types of method suffer from the ambiguity of human driving decisions, and further works have focused on increasing the interpretability of human data [22] [40].

Adversarial imitation learning is another approach that can be used to mimic human driving. The method uses a discriminator to evaluate the similarity of the output of its generator network to expert behaviour [51]. It has been adopted by Y. koeberle *et al.* to perform autonomous driving tasks in different scenarios, including lane merging and roundabout [28]. However, the driving policy produced using this method has limited

robustness, especially when faced with situations that are rare or unseen from the training data [42].

The last human driver imitation method that we list here is the direct derivation of human driving skills or driving styles. T. Li *et al.* uses fuzzy logic to model human driving skills, and J. Chen *et al.* uses a neuro-fuzzy structure to model human perception when driving [33] [9]. Furthermore, P. Hand *et al.* adopt game approaches to model different driving styles (aggressive, normal, conservative) and derive matching decision-making results [21]. This study is an early attempt to mimic specific driving styles rather than generic driving behaviour, but relies heavily on a pre-assumed categorization of driving styles. In addition, direct derivation methods may be less efficient and introduce high computational cost.

1.2.2 Human Driver Mimicking for Personalized AV Control

We then discuss the work on human driver mimicking for personalized AV control. Upon emergence of this idea, there were already plenty of research works that tried to imitate human driving, as listed in the previous sub-section. Therefore, existing approaches that work well are adapted to meet the new goal. For example, M. Kuderer *et al.* uses inverse reinforcement learning to produce personalized AV behaviour via a learned cost function [31]. Inevitably, this method inherits the limitation of inverse reinforcement learning, such that tasks with a long horizon cannot be handled well due to computational burden and the ambiguity of human actions. Another approach that has been adapted is the direct derivation method. D. Bolduc *et al.* extract specific parameters from driving data and perform adaptive cruise control accordingly [5]. Similarly, Y. Feng *et al.* designs a Support Vector Machine (SVM) personalized on the driving behaviour of a target driver and uses it to make lane change decisions [14].

1.2.3 Human Driver Mimicking: A Brief Summary

In summary, existing methods for mimicking human driver can be classified into the Artificial Intelligence (AI)-based and AI-free categories. The AI-based methods suffer from three major drawbacks. Knowing that obtaining a large dataset for a single driver could be prohibitively time consuming, the performance of the algorithm is highly affected by the training dataset and lacks robustness when encountering situations that are poorly represented by the expert demonstration. This becomes a major source of concern, as the algorithm may return unexpected control commands and hinders the safety of driving.

In addition, these approaches do not guarantee the satisfaction of constraints, including dynamic constraints and safety constraints, which is true even for well-represented scenarios. Finally, human expert actions can be ambiguous, making it hard to train a policy or derive a unique reward function. On the other hand, the AI-free methods (e.g., [33]) often depend explicitly or implicitly on some classification of driving skills or styles, but a good categorization scheme is hard to develop.

Finally, while this thesis focuses on increasing people’s trust towards AV by driving style mimicking, it is worth noting that some researches are starting to explore whether mimicry is the best approach. The work of M. Schrum *et al.* uses a neural network architecture to learn parameters that characterize the driving style of a driver, then adjust the level of aggression of the learned style to study user preferences. The authors argue that an AV that fully mimics one’s driving style is not always the best. Instead, the user’s personality traits and driving habits may influence their preference on an AV that mimics their driving style [47]. This argument is still under investigation and remains an open question for future researches.

1.3 Literature Review II

Vehicle motion planning is an extensively studied problem and can be addressed using a variety of methods. Existing methods for vehicle motion planning can be broken down into four categories: a) graph-search-based methods, b) sampling-based methods, c) interpolating curve methods, and d) numerical optimization [18].

In graph search-based methods, a graph is formed based on the discretization of a 2D map by connecting neighbouring regions with edges and the optimal path that connects the start and end regions is determined. Examples of graph-search-based methods include A* and Dijkstra’s algorithm [34] [41]. However, these methods are independent of vehicle dynamics and may generate trajectories that are not trackable by a vehicle. On the other hand, in sampling-based methods where the state space is randomly sampled to find a connected path, some non-holonomic constraints of vehicles such as the maximum turning radius can be taken into account [26]. An example of sampling-based methods is the Rapidly-Exploring Random Tree (RRT) planner [27]. In interpolating curve planners, different types of curve are used to connect way points on a map and to smooth the trajectories based on vehicle dynamics and ride comfort. The types of curves used include Clothoid curves, polynomial curves, Bezier curves, etc. [18]

Although sampling-based and interpolating curve methods attempt to consider vehicle dynamics when planning a trajectory, complex vehicle dynamics cannot be fully integrated

into the algorithm and feasibility is not fully guaranteed. To address this issue, numerical optimization is adopted, leading to a process called trajectory optimization. Optimization-based vehicle motion planning falls under the category of trajectory optimization for a control system subject to constraints. The optimality of a trajectory can be measured using predefined cost indices and is application dependent. Examples of constraints include feasibility according to system dynamics, control input constraints, and collision avoidance constraints.

By considering a [Discrete-Time \(DT\)](#) state space system model and optimizing the trajectory over a finite time horizon, the trajectory optimization problem is converted to a [Nonlinear Programming \(NLP\)](#) [43]. For vehicle motion planning and for many other applications, nonlinear system dynamics is needed to improve the accuracy of trajectories, which results in complex nonconvex optimization problems. In addition, certain input and state constraints, such as collision avoidance constraints, are also nonconvex and challenging to incorporate. To solve these problems, line search and trust-region methods are two commonly-used approaches. Both being numerical optimization strategies that solve nonconvex problems, line search methods determine a step along some descent direction, while trust-region methods find the best step within a region around the current point [39].

Examples of line-search methods include Quasi Newton methods and interior point methods that are implemented by the widely used solvers IPOPT and ForcesPro [39, 55, 62]. Trust region methods, such as [Sequential Quadratic Programming \(SQP\)](#), approximate and solve the original [NLP](#) while defining a trust region to control the accuracy of the approximations [8] [48]. Both types of method rely on local approximations of the system dynamics that result in errors. Large errors can deteriorate the convergence performance, resulting in slower convergence, convergence to a local optimum with lower quality, or even failure to converge.

Finally, for online vehicle motion planning, Model Predictive Control (MPC) is another common approach [56] [63]. This optimal control technique minimizes the cost of a predetermined objective function using future vehicle states over a receding horizon predicted from the current state and the system dynamics. A sequence of control inputs is generated, but only the input in the first step is executed, after which the prediction horizon is shifted by one step [49]. Being an online approach, it demands fast computation of the optimization problem and faces a trade-off between solution optimality and solving time.

1.4 Contributions

From the literature review on personalized **AV** controller in Chapter 1.2, we see that existing methods have the following major limitations: a) personalized based on user response or tuned with user inputs but ignore the user’s own driving style, b) **AI**-based methods lack robustness and safety guarantees and suffer from ambiguity of expert actions, c) direct derivation methods rely somewhat on pre-established classification. Therefore, our goal is to develop a personalized **AV** controller that a) mimics the user’s own driving style while ensuring safety, b) takes an explicit form and is robust to rare driving scenarios, and c) does not assume a driving style classification in advance.

To mimic human driving in an unambiguous way, we consider a parameterized closed-form cost function that is minimized during vehicle motion planning. The cost function is the sum of many cost metrics, such as metrics that penalize large control effort, high fuel consumption, etc. Each cost metric has a weight that determines its priority level when performing trajectory optimization. Various driving styles correspond to different combinations of weights, and these weights can be used as parameters to characterize a driver’s driving style.

To obtain the optimal weight combination for a specific driver, we minimize the difference between vehicle trajectories from the driver’s driving data and the vehicle motion planner. This suggests a bilevel controller formulation, whose **Lower-level (LL)** is a vehicle motion planner parameterized by weights, and **Upper-level (UL)** performs driving style imitation by tuning the weights to minimize the mismatch between imitated and true driving data. This design has the following benefits: a) mimics a user’s driving style based on their own driving data, b) the cost function for vehicle motion planning is closed-form, explicit, and easily interpretable with pre-selected cost metrics, c) the algorithm is robust to unseen driving scenarios, as the vehicle motion planner always produces a feasible, obstacle-free, safe trajectory, d) no driving style classification is assumed and all weight combinations are specific to test drivers.

When solving the **LL** vehicle trajectory optimization problem, we propose an input-to-state reformulation of the nonconvex trajectory optimization problem, which can be solved using **SQP** with constraint linearization based on trajectory sensitivities. **SQP** relies on a quadratic approximation of the nonconvex trajectory optimization problem at each iteration, which is obtained by linearizing the constraints. To do so, the dynamics constraints of the system are rewritten as a map from all control inputs to the full trajectory of system states. Then, this input-output map is linearized with respect to control inputs, resulting in what are known as trajectory sensitivities, and **SQP** is used to solve the resulting problem. Trajectory sensitivities are the derivatives of the system trajectory with respect to all

control inputs [23] [16]. Unlike derivatives with respect to states and inputs at each stage, trajectory sensitivities consider the initial state and the entire input trajectories, and do not depend on other states.

When constructing the quadratic approximation of the original problem, existing approaches typically rely on derivatives of each stage to effectively obtain a local linear approximation of the constraints at each iteration [39]. This often results in poor numerical approximation quality with accumulating numerical errors, as discussed in later sections of this thesis. Poor approximation quality can lead to reduced convergence rates or even failure to converge. We will show that trajectory sensitivities improve the numerical approximation accuracy of nonconvex trajectory optimization problems, and demonstrate that this leads to larger regions of convergence and improved convergence rates in practice.

Our contributions can be summarized as follows.

1. We propose a bilevel programming formulation of a personalized AV controller, which is able to mimic the driving style of a target driver with a closed-form cost function for vehicle motion planning, whose performance is robust to training data distribution, has improved safety guarantees under unseen driving scenarios, and has better generalizability to various situations.
2. We develop a novel linear approximation to nonlinear dynamics constraints in trajectory optimization problems by using an input-to-state map formulation combined with trajectory sensitivities. We show that this approximation results in improved numerical accuracy compared to traditional approaches.
3. We apply our novel linear approximation to develop a novel SQP-based method to solve nonconvex trajectory optimization problems with reduced numerical approximation errors, resulting in a larger region of convergence and faster convergence rates.
4. We provide local convergence guarantees for an important special case of the nonconvex trajectory optimization problem with input-affine inequality constraints using the proposed trajectory-sensitivities-based SQP approach.
5. To solve the driving style imitating problem, we implement a projected subgradient update method. We calculate derivatives of the LL objective with respect to UL weights by adapting a sensitivity analysis for the LL problem with input-to-state map and second-order trajectory sensitivities. These derivatives are then naturally used as descent directions to solve the outer problem. However, since the inner

problem is not differentiable everywhere, we instead employ projected subgradient update.

6. We test our vehicle motion planner by applying it to generate trajectories and compare the results with the traditional **SQP** method. We show that our proposed planner outperforms traditional **SQP** in convergence speed and robustness, even on reference trajectories that are not dynamically feasible, are nonsmooth, and pass through obstacle.
7. We also test our personalized **AV** controller using simulated driving data in different driving scenarios and examine how closely the resulting vehicle trajectories match the simulated ones. We show that our proposed algorithm can reproduce vehicle trajectories that are similar to expert demonstrations and hence, effectively imitate driving style.

1.5 Organization of the Thesis

The remainder of the thesis is organized as follows. Chapter 2 presents the problem formulation of the personalized **AV** controller and gives some technical preliminaries ; Chapter 3 discusses the vehicle motion planning problem; Chapter 4 develops an algorithm for solving the human driver mimicking and driving style imitating problem; Chapter 5 documents the simulation results of the two problems; Chapter 6 concludes the thesis, points out some limitations, and points to future research directions.

Chapter 2

Preliminaries

In this chapter, we show how the personalized AV controller is formulated into a bilevel structure and the subproblem that each level of the structure solves. Then, we present some technical preliminaries and existing results that become useful in later analysis.

2.1 Problem Statement

Our objective is to design a personalized AV controller that operates a vehicle to drive in a way similar to that of the user's own driving style. This controller should follow a closed-form cost function, be robust to unseen driving scenarios so as to perform well in unseen driving scenarios, maintain safety, and handle various types of driving styles.

2.2 Problem Formulation

A vehicle controller can be designed in a hierarchical structure, where a desired vehicle trajectory is planned first, and a tracking controller is used next to follow this trajectory. This hierarchical design has been shown to have good performance [38]. We want to determine off-line a closed form AV controller that isolates the computation burden from online implementation. Therefore, we focus on the planning phase when solving the personalized AV control problem. In addition, in later sections, we will discuss how part of our method could be adapted to form an MPC-based feedback controller to solve the tracking problem as well.

The basis of the proposed personalized AV controller is a vehicle motion planner that produces feasible and obstacle-free vehicle trajectories. A numerical optimization-based planner where vehicle dynamics are incorporated as constraints suits the need. This planner can then be modified to achieve driving habit mimicking and personalization. To do so, we parametrize the cost function of the trajectory optimization problem with weights of each cost metric as input parameters. The set of weights are determined such that the trajectories generated by the vehicle motion planner match those produced by the driver.

Given a set of weights that describe a driving style, we see that the personalized AV control problem described above can be divided into two subproblems. First, we need to generate vehicle trajectories using those weights, which we call the vehicle motion planning problem. Then, we need to adjust the weight combination to match the driving style of a user, which we refer to as the driving style imitating problem. We note that when solving the latter problem, we need to analyze vehicle trajectories generated using our planner with different weights and find the optimal one that best matches user’s data. This suggests that the vehicle motion planning problem is actually an inner layer of the driving style imitating problem. Nevertheless, this problem is self-contained and is of independent interest.

We note that the driving style imitating problem relies on human drivers’ driving data, which are either simulated or collected in real life and thus dynamically feasible. In contrast, the vehicle motion planning problem applies to references that are dynamically infeasible, nonsmooth, and that pass through obstacles. Eventually, we aim to find personalized, dynamically feasible paths given potentially infeasible reference paths.

2.2.1 Vehicle Motion Planning Problem

To begin, we assume that we are given a starting point of the vehicle, an end point, and a pre-planned path. This path can be generated using other simple motion planners and is a DT representation of vehicle states over time that connects the start and end points along the roads. This path is referred to as a "reference" path, which may not be feasible for a vehicle to track, or may pass through obstacles. The goal of our vehicle motion planner is to generate a dynamically feasible, obstacle-free and safe trajectory based on the given reference.

Consider the Continuous-Time (CT) vehicle dynamics in the following Ordinary Differential Equation (ODE) form:

$$\dot{x} = g'(x, u), \tag{2.1}$$

where $x \in \mathbb{R}^n$ is the state vector of the system, $u \in \mathbb{R}^m$ is the input vector, and n and m are the dimensions of the state and the input, respectively.

Discretize with sampling time T_s to obtain DT dynamics of the system.

$$x^{k+1} = g(x^k, u^k), \quad (2.2)$$

where k is the index of the time step, $g : \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^n$ is a vector of nonlinear functions that map the current state and input to the state in the next time step. This expression is used as equality constraints in trajectory optimization.

To implement the state and input constraints, we define the inequality:

$$h(x^k, u^k) \leq 0,$$

where $h : \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^p$ is also a possibly nonlinear vector of functions and p is the total number of inequality constraints. We assume that the constraints g and h are C^2 , and in general they are nonlinear and nonconvex.

Recall that x describes a collection of vehicle states. Depending on the vehicle model we choose, x may include positions, velocities, accelerations, etc. However, since each vehicle state x^k is tied to a time step k with sampling time T_s , the velocity and acceleration profiles are implicitly captured by the time series of positional information. Therefore, when approaching the driving style imitating problem, it suffices to match vehicle positions with driver's over a finite time horizon. To facilitate future analysis, define the output $y^k \in \mathbb{R}^2$ of the system that contains the X and Y positions of the vehicle in a fixed global coordinate frame of observation. Also, define matrix $C \in \mathbb{R}^{2 \times n}$ such that

$$y^k = Cx^k,$$

which exists given a Cartesian coordinate system of the vehicle model, where the x axis coincides with the longitudinal direction and the y axis aligns with the lateral direction. Such a choice of coordinates is common for vehicle models.

Over a finite time horizon of length T , we define the state and position trajectory of a dynamical system as

$$\mathbf{x} = [(x^0)^\top, \dots, (x^T)^\top]^\top, \quad \mathbf{y} = [(y^0)^\top, \dots, (y^T)^\top]^\top.$$

Now consider the DT trajectory optimization problem, which is formulated as an NLP

subject to equality and inequality constraints.

$$\begin{aligned}
& \min_{x^k, u^k} J_L(w, \mathbf{x}, \mathbf{u}, \mathbf{x}_r) \\
& \text{s.t. } x^{k+1} = g(x^k, u^k), \quad \forall k \in \{0, 1, \dots, T-1\}, \\
& \quad h(x^k, u^k) \leq 0, \quad \forall k \in \{0, 1, \dots, T-1\}, \\
& \quad y^k = Cx^k, \quad \forall k \in \{0, 1, \dots, T-1\}, \\
& \quad \mathbf{x} = [(x^0)^\top, \dots, (x^T)^\top]^\top, \quad \mathbf{u} = [(u^0)^\top, \dots, (u^T)^\top]^\top, \\
& \quad \mathbf{y} = [(y^0)^\top, \dots, (y^T)^\top]^\top,
\end{aligned} \tag{2.3}$$

where $x^k \in \mathbb{R}^n$ is the state at time k ; $u^k \in \mathbb{R}^m$ is the input at time k ; T is the length of the time horizon; $\mathbf{x} \in \mathbb{R}^{nT}$, $\mathbf{u} \in \mathbb{R}^{mT}$ and $\mathbf{y} \in \mathbb{R}^{2T}$ are the collections of all states, inputs, and positions over the time horizon T , respectively; $\mathbf{x}_r \in \mathbb{R}^{nT}$ is some constant reference state trajectory, which depends on the provided reference path; $J : \mathbb{R}^{nT \times mT} \rightarrow \mathbb{R}$ is the objective function; $w \in \mathbb{R}_{++}^l$ is the vector of l weight coefficients for motion planning; $g : \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^n$ is the nonlinear system dynamics; and $h : \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^p$ is the nonlinear element-wise inequality constraints.

In this work, we consider the setting in which the objective function J is quadratic, with which many practical objectives can be formulated. Let $w_1 \in \mathbb{R}^{l_1}$ and $w_2 \in \mathbb{R}^{l_2}$ be the weight vectors of states and inputs, respectively, and such that $w = [w_1 \ w_2]$. Then, the objective takes on the following form

$$J_L(w, \mathbf{x}, \mathbf{u}, \mathbf{x}_r) = \frac{1}{2}(\mathbf{x} - \mathbf{x}_r)^T P_1(w_1)(\mathbf{x} - \mathbf{x}_r) + \frac{1}{2}\mathbf{u}^T P_2(w_2)\mathbf{u},$$

where $P_1 \in \mathbb{R}^{nT \times nT}$ and $P_2 \in \mathbb{R}^{mT \times mT}$ are diagonal weight matrices constructed from the weight vector w_1 and w_2 .

Next, we define some functions that will be used in the problem formulation of the driving style imitating problem. First, we see that the cost function J is parameterized by the weight vector w , and changing w affects the solution of the optimization problem (2.3). In other words, the optimal input trajectory \mathbf{u}_* of (2.3) is a function of w . Define the weight-to-input mapping $F : \mathbb{R}^l \rightarrow \mathbb{R}^{mT}$:

$$\mathbf{u}_* = F(w, \mathbf{x}_r).$$

The reference trajectory is also included as it is an external input as well.

Then, we note that (2.2) is the deterministic dynamics of the vehicle, and that given an input trajectory \mathbf{u} , a state trajectory \mathbf{x} can be uniquely determined. Being parts of

vehicle states, the positional trajectory \mathbf{y} is also uniquely determined by \mathbf{u} . Define the input-to-position mapping $Y : \mathbb{R}^{mT} \rightarrow \mathbb{R}^{2T}$:

$$\mathbf{y} = Y(\mathbf{u}).$$

When solving each vehicle motion planning problem, the parameter w is fixed. Also, given that \mathbf{y} is defined to be used in the driving style imitating problem, the equality constraints of \mathbf{y} may be omitted. This leads to a simplified form of the problem:

$$\begin{aligned} \min_{x^k, u^k} \quad & J_L(\mathbf{x}, \mathbf{u}, w) \\ \text{s.t.} \quad & x^{k+1} = g(x^k, u^k), \quad \forall k \in \{0, 1, \dots, T-1\}, \\ & h(x^k, u^k) \leq 0, \quad \forall k \in \{0, 1, \dots, T-1\}, \\ & \mathbf{x} = [(x^0)^\top, \dots, (x^T)^\top]^\top, \quad \mathbf{u} = [(u^0)^\top, \dots, (u^T)^\top]^\top. \end{aligned} \tag{2.4}$$

This will be the problem formulation that we refer to when addressing the vehicle motion planning task alone.

2.2.2 Driving Style Imitating Problem

In the driving style imitating problem, we try to find the weight vector w with which the vehicle motion planner generates trajectories that match the driver's data. To do so, we require a number of demonstrations created by the target driver. Each demonstration should contain the following information:

- Initial vehicle states, $(x^0)_d$,
- The reference path, $(\mathbf{x}_r)_d$
- The vehicle path extracted from the target driver's driving data, \mathbf{y}_d ,

where the subscript d is the index of demonstration, and there is a total number of n_d demonstrations. We assume that the initial vehicle states coincide with the initial state of the reference path, which simplifies the formulation of the problem.

In addition to the desired path \mathbf{y}_d , we obtain another vehicle path \mathbf{y} by solving the vehicle motion planning problem with some weight vector w . In the driving style imitating

problem, our goal is to find w such that \mathbf{y}_d and \mathbf{y} match as closely as possible. This can be realized by minimizing the following cost function:

$$J_U(\mathbf{y}_d, \mathbf{u}_*, w) = \sum_{d=1}^{n_d} \|\mathbf{y}_d - \mathbf{y}(\mathbf{u}_*(w, (\mathbf{x}_r)_d))\|^2. \quad (2.5)$$

Here, the term $\mathbf{y}(\mathbf{u}_*)$ indicates that \mathbf{y} is a function of the optimal input trajectory \mathbf{u}_* obtained by solving the motion planning problem. This is equivalent to applying the input-to-position mapping Y to \mathbf{u}_* as $Y(\mathbf{u}_*)$.

Combining the vehicle motion planning problem with the driving style imitating problem, we formulate the **Bilevel Programming (BP)** problem:

$$\underset{w}{\text{minimize}} \quad J_U = \sum_{d=1}^{n_d} \|\mathbf{y}_d - Y(\mathbf{u}_*(w, (\mathbf{x}_r)_d))\|^2 \quad (2.6a)$$

$$\text{such that for each } d \in \{1, \dots, n_d\}: \quad (2.6b)$$

$$\mathbf{u}_* = \underset{\mathbf{u}}{\text{argmin}} \quad J_L(w, \mathbf{x}, \mathbf{u}, (\mathbf{x}_r)_d) \quad (2.6c)$$

$$\text{subject to} \quad x^{k+1} = x^k + g(x^k, u^k), \quad k \in \{0, \dots, T-1\}, \quad (2.6d)$$

$$h(x^k, u^k) \leq 0, \quad k \in \{0, \dots, T-1\}, \quad (2.6e)$$

$$y^k = Cx^k, \quad k \in \{0, \dots, T-1\}, \quad (2.6f)$$

$$x^0 = (x^0)_d, \quad (2.6g)$$

$$\mathbf{x} = [(x^0)^\top, \dots, (x^T)^\top]^\top, \quad \mathbf{u} = [(u^0)^\top, \dots, (u^T)^\top]^\top, \quad (2.6h)$$

$$\mathbf{y} = [(y^0)^\top, \dots, (y^T)^\top]^\top. \quad (2.6i)$$

Solving this problem, we get the optimal weight combination w_* that best characterizes a user's driving style. Now, whenever we solve the motion planning problem again, even for new scenarios, with w_* , we will get a personalized vehicle trajectory, tracking which the vehicle is controlled in a personalized fashion.

In the problem formulation in (2.6), a common length of the prediction horizon, T , is used. To handle varying lengths of prediction horizons, we can append the length of the horizon for a demonstration, T_d , to each expert data and include it as an input to the **LL** objective function J_L .

Using the input-to-weight map F , the problem can be simplified to:

$$\begin{aligned}
& \min_w J_U(\mathbf{y}_d, \mathbf{y}(\mathbf{u}_*(w, (\mathbf{x}_r)_d))) \\
& \text{s.t. for each } d \in \{1, \dots, n_d\}: \\
& \quad \mathbf{u}_* = F(w, (\mathbf{x}_r)_d), \\
& \quad w \geq 0.
\end{aligned} \tag{2.7}$$

2.3 Key Results from Related Work

In this section, we summarize some established results and key findings from existing works that are relevant to our work.

2.3.1 Sequential Quadratic Programming

SQP is a popular method for solving nonconvex optimization problems. The textbook *Numerical Optimization* has a chapter detailing this method. Using **SQP**, we approximate the original problem as **Quadratic Programming (QP)** subproblems and solve them iteratively until convergence. Conventionally, quadratic approximations are formulated by linearizing the constraints [39]. Take the equality constraints in Equation (2.3) as examples, they are expressed as

$$x_{s+1}^{k+1} = x_s^{k+1} + \frac{\partial g}{\partial x}(x_{s+1}^k - x_s^k) + \frac{\partial g}{\partial u}(u_{s+1}^k - u_s^k)$$

for every time step k and each iteration s . The multistage nature of these linearizations often lead to accumulation of numerical error as k grows larger, which will be elaborated in later sections.

To limit the magnitude of approximation errors, trust-region constraints are added to the subproblems. In situations where the subproblem is infeasible, the hard constraints can be relaxed to soft constraints by adding relaxation factors that are also introduced in the cost function with increasing weights. Being a trust region method, **SQP** allows us to use the size of the trust region as a metric to quantify and compare the local approximation accuracy.

The convergence of **SQP** methods is typically established using a merit function. One typical choice is the weighted sum of the original objective and the amounts of constraint violation. This strategy then requires additional assumptions to ensure convergence, which we omit here as they are not related to this work.

2.3.2 Tensor Operation, Partial Derivatives, and Einstein Summation Notation

Since we are working with a dynamical system with multiple states and inputs, the second-order derivatives of the system becomes a tensor, and any relevant computation becomes tensor operations. Einstein summation notation improves clarity and brevity when manipulating multidimensional quantities. Here we introduce the Einstein summation notation and give a simple example when using it in derivative computation [2] [16].

Given an expression with superscripts and subscripts, Einstein summation notation implicitly suggests that terms with an index that only appears on one side of an equation should be summed over that index from 1 to N , where N is the dimensionality of the space. For example, the following equation is written in Einstein summation notation:

$$a^i = f_\alpha^i b^\alpha,$$

which is equivalent to

$$a^i = \sum_\alpha f_\alpha^i b^\alpha.$$

Here, α is the repeated index that only appears on the right-hand side of the expression. The other index, i , appears on both sides of the expression and is not summed over.

We combine the use of Einstein summation notation with a simplified notation of partial derivatives. Suppose that we are taking the derivatives of an expression with respect to a vector y , and let the subscript j denote the j th component of y . Then, the partial derivative of a vector b with respect to y_i is denoted by

$$\frac{\partial b}{\partial y_j} =: b_j.$$

Combining this with Einstein summation notation, when seeing a repeated subscript, the object with respect to which we take the derivative of is identified by the term with the same index. In other words, the expression

$$a_j^i = f_\alpha^i b_j^\alpha$$

is equivalent to

$$\frac{\partial a^i}{\partial y_j} = \sum_\alpha \frac{\partial f^i}{\partial b_\alpha} \frac{\partial b_\alpha}{\partial y_j},$$

which is the result of applying the chain rule.

To avoid ambiguity, we use the Einstein summation notation when working with second-order derivatives. For first-order derivatives, we use the usual fractional notation to improve readability.

2.3.3 Trajectory Sensitivity

To address the numerical issues of local approximation in traditional SQP, we leverage the concept of trajectory sensitivities. Trajectory sensitivities are the derivatives of the states of the system with respect to the input trajectory at each time step. Using trajectory sensitivities, it is possible to linearize the dynamics of a system about some nominal trajectory rather than about a point of equilibrium [23].

The work of M. W. Fisher and I. A. Hiskens shows how first-, second- and third-order trajectory sensitivities of a dynamical system in ODE form can be computed [16]. Extending to the DT scenario, consider a simple example where p is the input, then the first-order trajectory sensitivity in time step k is $\frac{dx^k}{dp}$. Stacking the trajectory sensitivities in all time steps along the finite time horizon, the full trajectory sensitivities $\frac{dx}{dp}$ can be obtained.

2.3.4 Subgradient and Subdifferentials

We use a gradient-based method to solve the BP problem. When a function is differentiable, the gradient at a point is unique and the gradient computation is straightforward. However, for a BP problem, the function that we try to differentiate is a composition of the UL objective and the nonconvex LL optimization problem as:

$$J_H(\mathbf{y}_d, \mathbf{y}(\mathbf{u}_*(w, (\mathbf{x}_r)_d))) = J_H(\mathbf{y}_d, \mathbf{y}(F(w, (\mathbf{x}_r)_d))).$$

In this case, differentiability generally does not hold. Instead, we use the notion of subgradients. Given a convex function f , we say that $g \in \mathbb{R}^n$ is the subgradient of f at a point $x \in \mathbb{R}^n$ if

$$f(y) \geq f(x) + \langle g, y - x \rangle \quad \forall y \in \mathbb{R}^n.$$

The subgradients are not always unique. Then, we define the collection of subgradients to be the subdifferential, as

$$\partial f = \{g \in \mathbb{R}^n : f(y) \geq f(x) + \langle g, y - x \rangle \quad \forall y \in \mathbb{R}^n\}.$$

When the function f is differentiable at x , the subdifferential is a singleton and equals to the gradient.

When f is nonconvex, the Clark Jacobian can be used as a generalization of the subdifferential, which is defined as

$$\mathcal{J}^c f = \text{conv}\{\lim_{k \rightarrow \infty} \partial f(x^k) \mid x^k \rightarrow x, x^k \in \Omega\},$$

where $\Omega \subseteq \mathbb{R}^n$ is the set of points where f is differentiable [11]. In other words, the Clark Jacobian equals to the subgradient when f is differentiable.

2.3.5 Additional Notation

A matrix-induced inner product is defined as $\langle \mathbf{x}, \mathbf{x} \rangle_A = \mathbf{x}^\top A \mathbf{x}$ where A is a symmetric positive definite matrix. This inner product can be used to measure distance. We use the symbol $\|\cdot\|$ to represent a norm, and the symbols $B_r(p)$ and $\overline{B}_r(p)$ to denote the open ball and the closed ball of radius r around the point p .

Chapter 3

Motion Planning Problem

In this chapter, we present a novel SQP-based approach to solve the LL vehicle motion planning problem using trajectory optimization. In traditional SQP method, nonlinear dynamics constraints are linearized using first-order Taylor's approximation, which results in cumulative numeric error. We propose using Trajectory Sensitivities (TS) to perform constraint linearization, which is not affected by this type of error. We also present a convergence analysis of the proposed method for an important special class of nonconvex trajectory optimization problems.

3.1 Linearization of System Dynamics

When using SQP methods to solve nonconvex optimization problems, the original problem is approximated by quadratic subproblems and solved iteratively. To obtain subproblems of the original problem, the constraints need to be linearized. In this section, we discuss how constraint linearization can be achieved using an input-to-state map of vehicle dynamics and a TS-based approximation. We also compare the approximation results using first-order Taylor series with those using trajectory sensitivities.

3.1.1 Input-to-state map

First, we perform an input-to-state reformulation of the problem in (2.3). Given some initial state x^0 , all subsequent states x^1, x^2, \dots, x^{T-1} along the time horizon depend on the

inputs u^1, u^2, \dots, u^{T-1} . The collection of states, \mathbf{x} , is the trajectory of the dynamical system and is uniquely determined by x^0 and \mathbf{u} . This input-to-state map can be described by using

$$G(x^0, \mathbf{u}) = \mathbf{x}, \quad (3.1)$$

where $G : \mathbb{R}^{n \times mT} \rightarrow \mathbb{R}^{nT}$ is a nonlinear and nonconvex reformulation of the system dynamics. A similar input-to-state reformulation applied to inequality constraints can be written as $H(x^0, \mathbf{u}) \leq 0$, where $H : \mathbb{R}^{n \times mT} \rightarrow \mathbb{R}^p$, and are element-wise constraints. Here we use the fact that, based on Equation (3.1), \mathbf{x} can be expressed as a function of \mathbf{u} alone. Note that G and H are defined implicitly and that explicit formulas for them are typically difficult to determine and are not required for our proposed approach. In addition, for the CT dynamical system defined in Equation (2.1), if we assume that the vector field is at least locally Lipschitz, then uniqueness is guaranteed and we can define uniquely the maps G and H .

We re-express the objective in quadratic form as

$$J_L(\mathbf{x}, \mathbf{u}) = \frac{1}{2} \left(\begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix} - \begin{bmatrix} \mathbf{x}_r \\ \mathbf{u}_r \end{bmatrix} \right)^\top \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} \left(\begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix} - \begin{bmatrix} \mathbf{x}_r \\ \mathbf{u}_r \end{bmatrix} \right), \quad (3.2)$$

where $Q \in \mathbb{R}^{nT \times nT}$ and $R \in \mathbb{R}^{mT \times mT}$ are positive definite weight matrices, and $\mathbf{x}_r \in \mathbb{R}^{nT}$ and $\mathbf{u}_r \in \mathbb{R}^{mT}$ are some constant vectors.

Combining the above reformulations, the trajectory optimization NLP takes on a new form

$$\begin{aligned} \min_{\mathbf{x}, \mathbf{u}} \quad & J_L(\mathbf{x}, \mathbf{u}) \\ \text{s.t.} \quad & G(x^0, \mathbf{u}) = \mathbf{x}, \\ & H(x^0, \mathbf{u}) \leq 0. \end{aligned} \quad (3.3)$$

To solve Equation (3.3), we propose a variant of SQP that linearizes the constraints using first-order trajectory sensitivities, which makes it possible for us to accurately compute the derivative of the implicit input-to-state map G with respect to the input trajectory \mathbf{u} .

3.1.2 Trajectory Sensitivities Computation

Let \mathbf{u} be the trajectory of all inputs to the system along the prediction horizon. Using system dynamics in (2.3), first-order trajectory sensitivity at time step $k+1$ can be efficiently computed as

$$\left. \frac{dx}{d\mathbf{u}} \right|_{k+1} = \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{dx}{d\mathbf{u}} \right|_k + \left. \frac{\partial g}{\partial \mathbf{u}} \right|_k. \quad (3.4)$$

The full trajectory sensitivities are evaluated in an iterative fashion along the trajectory of the underlying dynamical system.

Similarly, the trajectory sensitivity of inequality constraints at time step $k + 1$ can be computed as

$$\left. \frac{dh}{d\mathbf{u}} \right|_{k+1} = \left. \frac{\partial h}{\partial x} \right|_k \left. \frac{dx}{d\mathbf{u}} \right|_k + \left. \frac{\partial h}{\partial \mathbf{u}} \right|_k. \quad (3.5)$$

We would like to derive an expression of trajectory sensitivities in terms of past trajectory sensitivities and other partial derivatives, which enables us to analyze the approximation results. To do so, we first initialize trajectory sensitivities to be

$$\left. \frac{dx}{d\mathbf{u}} \right|_0 = 0,$$

as the initial condition is fixed in our problem setup.

Then, we expand (3.4).

$$\begin{aligned} \left. \frac{dx}{d\mathbf{u}} \right|_{k+1} &= \left. \frac{\partial g}{\partial x} \right|_k \left(\left. \frac{\partial g}{\partial x} \right|_{k-1} \left. \frac{dx}{d\mathbf{u}} \right|_{k-1} + \left. \frac{\partial g}{\partial \mathbf{u}} \right|_{k-1} \right) + \left. \frac{\partial g}{\partial \mathbf{u}} \right|_k \\ &= \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial x} \right|_{k-1} \left. \frac{dx}{d\mathbf{u}} \right|_{k-1} + \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial \mathbf{u}} \right|_{k-1} + \left. \frac{\partial g}{\partial \mathbf{u}} \right|_k \\ &= \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial x} \right|_{k-1} \left(\left. \frac{\partial g}{\partial x} \right|_{k-2} \left. \frac{dx}{d\mathbf{u}} \right|_{k-2} + \left. \frac{\partial g}{\partial \mathbf{u}} \right|_{k-2} \right) + \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial \mathbf{u}} \right|_{k-1} + \left. \frac{\partial g}{\partial \mathbf{u}} \right|_k \\ &= \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial x} \right|_{k-1} \left. \frac{\partial g}{\partial x} \right|_{k-2} \left. \frac{dx}{d\mathbf{u}} \right|_{k-2} + \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial x} \right|_{k-1} \left. \frac{\partial g}{\partial \mathbf{u}} \right|_{k-2} + \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial \mathbf{u}} \right|_{k-1} + \left. \frac{\partial g}{\partial \mathbf{u}} \right|_k, \\ &= \dots \end{aligned}$$

and eventually get the following general expression.

$$\left. \frac{dx}{d\mathbf{u}} \right|_{k+1} = \sum_{i=0}^k \left[\left(\prod_{j=0}^{k-i-1} \left. \frac{\partial g}{\partial x} \right|_{k-j} \right) \left. \frac{\partial g}{\partial \mathbf{u}} \right|_i \right] \in \mathbb{R}^{n \times mT}. \quad (3.6)$$

In addition, looking at the dynamics described in (2.2), we see that at time step $k + 1$,

g is a function of u^k alone. This suggests that

$$\begin{aligned} \left. \frac{\partial g}{\partial \mathbf{u}} \right|_0 &= \left[\left. \frac{\partial g}{\partial u^0} \right|_0 \quad \left. \frac{\partial g}{\partial u^1} \right|_0 \quad \cdots \quad \left. \frac{\partial g}{\partial u^{T-1}} \right|_0 \right] = \left[\left. \frac{\partial g}{\partial u} \right|_0 \quad 0 \quad \cdots \quad 0 \right], \\ \left. \frac{\partial g}{\partial \mathbf{u}} \right|_1 &= \left[\left. \frac{\partial g}{\partial u^0} \right|_1 \quad \left. \frac{\partial g}{\partial u^1} \right|_1 \quad \cdots \quad \left. \frac{\partial g}{\partial u^{T-1}} \right|_1 \right] = \left[0 \quad \left. \frac{\partial g}{\partial u} \right|_1 \quad 0 \quad \cdots \quad 0 \right], \\ \left. \frac{\partial g}{\partial \mathbf{u}} \right|_2 &= \left[0 \quad 0 \quad \left. \frac{\partial g}{\partial u} \right|_2 \quad 0 \quad \cdots \quad 0 \right], \\ &\vdots \\ \left. \frac{\partial g}{\partial \mathbf{u}} \right|_k &= \left[\cdots \quad \left. \frac{\partial g}{\partial u} \right|_k \quad \cdots \right], \text{ where the } (k+1)\text{-th entry is non-zero.} \end{aligned}$$

Substituting these into equation (3.6) and denote the full trajectory sensitivities by $\frac{dG}{d\mathbf{u}}$, we can write

$$\begin{aligned} \frac{dG}{d\mathbf{u}} &:= \begin{bmatrix} \frac{dG_0}{d\mathbf{u}} \\ \frac{dG_1}{d\mathbf{u}} \\ \frac{dG_2}{d\mathbf{u}} \\ \vdots \\ \frac{dG_{T-1}}{d\mathbf{u}} \end{bmatrix} = \begin{bmatrix} \left. \frac{dx}{d\mathbf{u}} \right|_0 \\ \left. \frac{dx}{d\mathbf{u}} \right|_1 \\ \left. \frac{dx}{d\mathbf{u}} \right|_2 \\ \vdots \\ \left. \frac{dx}{d\mathbf{u}} \right|_{T-1} \end{bmatrix} \in \mathbb{R}^{nT \times mT} \\ &= \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ \frac{\partial g}{\partial u} \Big|_0 & 0 & \cdots & 0 & 0 \\ \frac{\partial g}{\partial x} \Big|_1 \quad \frac{\partial g}{\partial u} \Big|_0 & \frac{\partial g}{\partial u} \Big|_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \left(\prod_{j=0}^{T-2} \frac{\partial g}{\partial x} \Big|_{k-j} \right) \frac{\partial g}{\partial u} \Big|_0 & \left(\prod_{j=0}^{T-3} \frac{\partial g}{\partial x} \Big|_{k-j} \right) \frac{\partial g}{\partial u} \Big|_1 & \cdots & \frac{\partial g}{\partial u} \Big|_{T-2} & 0 \end{bmatrix}, \end{aligned} \quad (3.7)$$

where the row of zeros suggests fixed initial system states, and all other zero entries are consistent with the causality of the system.

3.1.3 TS-based linearization

Trajectory sensitivities arise naturally in the linearization of the input-to-state map G and H , using which system dynamics can be linearized. Recall that the proposed method is a variation of the **SQP** methods, which run in an iterative fashion. Rearrange equation (3.1) to get

$$\mathbf{x} - G(x^0, \mathbf{u}) = 0,$$

then linearize this expression in iteration s to get

$$\mathbf{x}_s - G(x^0, \mathbf{u}_s) + \mathbf{x}_{s+1} - \mathbf{x}_s - \left. \frac{dG}{d\mathbf{u}} \right|_s (\mathbf{u}_{s+1} - \mathbf{u}_s) = 0,$$

which can be simplified to

$$\mathbf{x}_{s+1} = \mathbf{x}_s + \left. \frac{dG}{d\mathbf{u}} \right|_s (\mathbf{u}_{s+1} - \mathbf{u}_s), \quad (3.8)$$

where $\left. \frac{dG}{d\mathbf{u}} \right|_s \in \mathbb{R}^{nT \times mT}$ is the first-order trajectory sensitivities, which can be computed using Equation (3.4) and (3.7). Here, the increment in the subscript represents a "new" or "approximated" state.

To perform a trajectory-sensitivities-based linearization of H , first define

$$H(x^0, \mathbf{u}) = \mathbf{h},$$

where $\mathbf{h} \in \mathbb{R}^p$ is a vector whose entries correspond to the values of H . Then, inequality constraints can be linearized using trajectory sensitivities as

$$\mathbf{h}_s + \left. \frac{dH}{d\mathbf{u}} \right|_s (\mathbf{u}_{s+1} - \mathbf{u}_s) \leq 0, \quad (3.9)$$

where first-order trajectory sensitivities can be computed using Equation (3.5).

Note that the linearization in Equations (3.8) and (3.9) is with respect to \mathbf{u} alone; hence the value of any state x^k depends only on the initial state and inputs. However, in the multistage approximation, each state depends on the previous state, which allows the numerical error to propagate through. Therefore, trajectory-sensitivities-based linearization has fewer approximation errors caused by numerical issues. This will be discussed in more detail in later sections.

3.1.4 Comparison of Approximation Results

In this section, we perform a theoretical comparison of linearization results using the first-order Taylor approximation and the TS-based approximation. The original system dynamics at iteration number s is

$$x_s^{k+1} - g(x_s^k, u_s^k) = 0, \quad \forall k \in \{0, \dots, T-1\}. \quad (3.10)$$

We first consider the linearization of the system dynamics using the first-order Taylor approximation. Define $\Delta x_s^k = x_{s+1}^k - x_s^k$ and $\Delta u_s^k = u_{s+1}^k - u_s^k$; then, we linearize (3.10) as

$$\Delta x_s^{k+1} = \left. \frac{\partial g}{\partial x} \right|_k \Delta x_s^k + \left. \frac{\partial g}{\partial u} \right|_k \Delta u_s^k,$$

which is equivalent to

$$x_{s+1}^{k+1} = x_s^{k+1} + \left. \frac{\partial g}{\partial x} \right|_k \Delta x_s^k + \left. \frac{\partial g}{\partial u} \right|_k \Delta u_s^k. \quad (3.11)$$

Recall that the state trajectory is fully determined by the input trajectory, \mathbf{u} , and that TS-based linearization is a function of the change in \mathbf{u} . For consistency, we expand (3.11) to obtain an expression of Δu as well.

$$\begin{aligned} x_{s+1}^{k+1} &= x_s^{k+1} + \left. \frac{\partial g}{\partial x} \right|_k \left(\left. \frac{\partial g}{\partial x} \right|_{k-1} \Delta x_s^{k-1} + \left. \frac{\partial g}{\partial u} \right|_{k-1} \Delta u_s^{k-1} \right) + \left. \frac{\partial g}{\partial u} \right|_k \Delta u_s^k \\ &= x_s^{k+1} + \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial x} \right|_{k-1} \Delta x_s^{k-1} + \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial u} \right|_{k-1} \Delta u_s^{k-1} + \left. \frac{\partial g}{\partial u} \right|_k \Delta u_s^k \\ &= x_s^{k+1} + \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial x} \right|_{k-1} \left(\left. \frac{\partial g}{\partial x} \right|_{k-2} \Delta x_s^{k-2} + \left. \frac{\partial g}{\partial u} \right|_{k-2} \Delta u_s^{k-2} \right) \\ &\quad + \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial u} \right|_{k-1} \Delta u_s^{k-1} + \left. \frac{\partial g}{\partial u} \right|_k \Delta u_s^k \\ &= x_s^{k+1} + \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial x} \right|_{k-1} \left. \frac{\partial g}{\partial x} \right|_{k-2} \Delta x_s^{k-2} + \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial x} \right|_{k-1} \left. \frac{\partial g}{\partial u} \right|_{k-2} \Delta u_s^{k-2} \\ &\quad + \left. \frac{\partial g}{\partial x} \right|_k \left. \frac{\partial g}{\partial u} \right|_{k-1} \Delta u_s^{k-1} + \left. \frac{\partial g}{\partial u} \right|_k \Delta u_s^k \\ &= \dots \end{aligned}$$

which can be written compactly as

$$x_{s+1}^{k+1} = x_s^{k+1} + \sum_{i=0}^k \left[\left(\prod_{j=0}^{k-i-1} \left. \frac{\partial g}{\partial x} \right|_{k-j} \right) \left. \frac{\partial g}{\partial u} \right|_i \Delta u_s^i \right]. \quad (3.12)$$

We now consider the TS-based linearization of the system dynamics, which has been discussed in earlier sections. Combining equations (3.7) and (3.8), and expanding the state

trajectory, \mathbf{x} , we can write

$$\begin{bmatrix} x_{s+1}^0 \\ x_{s+1}^1 \\ x_{s+1}^2 \\ \vdots \\ x_{s+1}^{T-1} \end{bmatrix} = \begin{bmatrix} x_s^0 \\ x_s^1 \\ x_s^2 \\ \vdots \\ x_s^{T-1} \end{bmatrix} + \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ \frac{\partial g}{\partial u}|_0 & 0 & \dots & 0 & 0 \\ \frac{\partial g}{\partial x}|_1 \frac{\partial g}{\partial u}|_0 & \frac{\partial g}{\partial u}|_1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \left(\prod_{j=0}^{T-2} \frac{\partial g}{\partial x}|_{k-j}\right) \frac{\partial g}{\partial \mathbf{u}}|_0 & \left(\prod_{j=0}^{T-3} \frac{\partial g}{\partial x}|_{k-j}\right) \frac{\partial g}{\partial \mathbf{u}}|_1 & \dots & \frac{\partial g}{\partial \mathbf{u}}|_{T-2} & 0 \end{bmatrix} \begin{bmatrix} \Delta u_s^0 \\ \Delta u_s^1 \\ \Delta u_s^2 \\ \vdots \\ \Delta u_s^{T-1} \end{bmatrix}.$$

Looking at the $(k+2)$ -th row of the matrix equality, we get an expression of x_{s+1}^{k+1} , which is identical to the expression in (3.12) if we expand the matrix multiplication on the right-hand side of the equality.

We see that the estimated states \mathbf{x}_{s+1} using both approximation methods seem identical. However, we still favor TS-based linearization more than the first-order Taylor approximation, since the latter method suffers from cumulative numerical errors when implemented in an optimization problem. This will be explained in further detail in the next section.

3.2 Modified SQP Method

We now propose an SQP-based algorithm to solve nonconvex optimization problems, and compare how the two different types of approximation (Taylor's series approximation and TS-based input-to-state approximation) affect estimation accuracy of the new state trajectory.

3.2.1 Algorithm Design

Linearization of constraints introduces approximation errors. To effectively limit the magnitude of these errors, a common method is to introduce a trust-region [39]. To do so, a suitable trust region radius Δ is selected to construct a constraint of the form

$$\|\mathbf{u}_{s+1} - \mathbf{u}_s\| \leq \Delta,$$

and define $d\mathbf{u} \in \mathbb{R}^{mT} := \mathbf{u}_{s+1} - \mathbf{u}_s$ for simplicity.

Combining the aforementioned modifications to the traditional SQP algorithm, a new variant of SQP with trajectory-sensitivities-based constraint linearization (SQP-TS) is pro-

posed. This method solves a series of quadratic subproblems of the form

$$\begin{aligned}
& \min_{d\mathbf{u}} && J_L(w, \mathbf{x}_{s+1}, \mathbf{u}_{s+1}, (\mathbf{x}_r)_d) \\
& \text{s.t.} && \mathbf{x}_s + \left. \frac{dG}{d\mathbf{u}} \right|_s d\mathbf{u} = \mathbf{x}_{s+1}, \\
& && \mathbf{h}_s + \left. \frac{dH}{d\mathbf{u}} \right|_s d\mathbf{u} \leq 0, \\
& && \mathbf{u}_{s+1} = \mathbf{u}_s + d\mathbf{u}, \\
& && \|d\mathbf{u}\| \leq \Delta,
\end{aligned} \tag{3.13}$$

until $\|d\mathbf{u}\|$ approaches zero, and the algorithm converges. Note that \mathbf{x}_s , \mathbf{u}_s , and \mathbf{h}_s are all constant vectors in the optimization problem at each iteration s , and the only decision variables are $d\mathbf{u}$.

The pseudocode of **SQP-TS** is presented in Algorithm 1. In the algorithm, we compute the exact state trajectory of the system, \mathbf{x}_s , on line 3. Then, we compute the trajectory sensitivities on line 4. On line 5, we construct linear approximations of constraints. On line 6, we solve the quadratic subproblem. Finally, on lines 7 and 8, we update the new inputs \mathbf{u}_{s+1} . Convergence is assessed by comparing the change between iterations of a set of selected system variables to a predefined tolerance.

Algorithm 1 SQP-TS

- 1: Choose initial trajectory $(\mathbf{x}_0, \mathbf{u}_0)$; set $s \leftarrow 0$
 - 2: **while** Convergence test not satisfied **do**
 - 3: Recompute the state trajectory \mathbf{x}_s by numerical iteration of the system dynamics in (2.3)
 - 4: Compute the trajectory sensitivities by iteration of (3.4), then evaluate (3.5), at \mathbf{u}_s
 - 5: Linearize the constraints using the trajectory sensitivities by (3.8) and (3.9)
 - 6: Solve the quadratic program (3.13) to obtain $d\mathbf{u}$
 - 7: Set $\mathbf{u}_{s+1} \leftarrow \mathbf{u}_s + d\mathbf{u}$ and $s \leftarrow s + 1$
 - 8: **end while**
-

3.2.2 Comparison of Approximation Error

Looking at the **QP** problem (3.13), the linearized system dynamics is included as equality constraints. When the optimization problem is solved, the equality constraints are usually

solved to some tolerances, resulting in small numeric errors in the prediction of new system dynamics. Denoting these errors by $\epsilon \in \mathbb{R}^{nT}$, the system dynamics approximated using trajectory sensitivities takes the form of

$$\mathbf{x}_{s+1} = \mathbf{x}_s + \left. \frac{dG}{d\mathbf{u}} \right|_s d\mathbf{u} + \epsilon. \quad (3.14)$$

For a single time step of $k + 1$, we get

$$x_{s+1}^{k+1} = x_s^k + \left(\left. \frac{dG}{d\mathbf{u}} \right|_s d\mathbf{u} \right)^k + \epsilon^{k+1}, \quad (3.15)$$

where $\epsilon^{k+1} \in \mathbb{R}^n$ is the numeric error associated with this equality when solving the optimization problem.

Now we consider the case where the system dynamics is linearized using Taylor's first-order approximation. In the $(k + 1)$ -th time step, we have:

$$x_{s+1}^{k+1} \approx x_s^{k+1} + \left. \frac{\partial g}{\partial x} \right|_k \Delta x_s^k + \left. \frac{\partial g}{\partial u} \right|_k \Delta u_s^k + \epsilon^{k+1},$$

However, we also note that $\Delta x_s^k = x_{s+1}^k - x_s^k$ where x_{s+1}^k is also approximated and comes with some numeric error ϵ^k . The expression of x_{s+1}^{k+1} then becomes

$$x_{s+1}^{k+1} \approx x_s^{k+1} + \left. \frac{\partial g}{\partial x} \right|_k \Delta x_s^k + \left. \frac{\partial g}{\partial u} \right|_k \Delta u_s^k + \epsilon^{k+1} + \left. \frac{\partial g}{\partial x} \right|_k \epsilon^k.$$

Repeating the same analysis for system states prior to x_{s+1}^k , the numeric errors are cumulated as we move along the prediction horizon. Compactly, we can write

$$x_{s+1}^{k+1} = x_s^{k+1} + \left. \frac{\partial g}{\partial x} \right|_k \Delta x_s^k + \left. \frac{\partial g}{\partial u} \right|_k \Delta u_s^k + \sum_{i=0}^k \left(\prod_{j=0}^{k-i-1} \left. \frac{\partial g}{\partial x} \right|_j \right) \epsilon^{i+1}. \quad (3.16)$$

The existence of cumulative errors in the Taylor first-order approximation shown in (3.16) makes the linearization of system dynamics less reliable and undermines the convergence performance of the algorithm. In contrast, the TS-based linearization only has a one-time numeric error as shown in (3.15) and produces better approximation results.

3.3 Convergence Analysis

Here we present a convergence analysis of the proposed SQP-TS algorithm. To begin, consider a local minimum of the NLP problem (3.3) at $(\mathbf{x}^*, \mathbf{u}^*)$. Let \bar{H} be the subset of H consisting of the active constraints at $(\mathbf{x}^*, \mathbf{u}^*)$, and define

$$\bar{G}(\mathbf{x}, \mathbf{u}) = [(\mathbf{x} - G(x^0, \mathbf{u}))^\top \quad \bar{H}(x^0, \mathbf{u})^\top]^\top \quad (3.17)$$

as the union set of equality and active inequality constraints consisting of \bar{n} elements.

Assumption 1. $d\bar{G}$ is full rank at $(\mathbf{x}^*, \mathbf{u}^*)$. Also known as Linear Independence Constraint Qualification (LICQ), this holds almost always by Sard's Theorem [39] [32].

We define $\mathbf{q} = [\mathbf{x}^\top \quad \mathbf{u}^\top]^\top$, $\mathbf{q}_r = [\mathbf{x}_r^\top \quad \mathbf{u}_r^\top]^\top$, and P a block diagonal matrix defined as $P = \text{diag}(Q, R)$.

By removing the inactive inequality constraints and considering the active inequality constraints as equality constraints, the original problem in (3.3) is equivalent to

$$\begin{aligned} \min_{\mathbf{q}} \quad & \frac{1}{2}(\mathbf{q} - \mathbf{q}_r)^\top P(\mathbf{q} - \mathbf{q}_r) \\ \text{s.t.} \quad & \bar{G}(\mathbf{q}) = 0. \end{aligned} \quad (3.18)$$

By removing the trust region constraints in (3.13), we get

$$\begin{aligned} \min_{d\mathbf{u}} \quad & J_L(w, \mathbf{x}_{s+1}, \mathbf{u}_{s+1}, (\mathbf{x}_r)_d) \\ \text{s.t.} \quad & \mathbf{x}_s + \left. \frac{dG}{d\mathbf{u}} \right|_s d\mathbf{u} = \mathbf{x}_{s+1}, \\ & \mathbf{h}_s + \left. \frac{dH}{d\mathbf{u}} \right|_s d\mathbf{u}_s \leq 0, \\ & \mathbf{u}_{s+1} = \mathbf{u}_s + d\mathbf{u}_s. \end{aligned} \quad (3.19)$$

The following result from [44] shows that the same constraints are active locally when making our approximation.

Lemma 1. *Under the assumptions of LICQ, strict complementarity condition, and second-order sufficient conditions, if \mathbf{q} is sufficiently close to $\mathbf{q}^* = (\mathbf{x}^*, \mathbf{u}^*)$, there is a solution of the subproblem (3.19) whose set of active inequality constraints is the same as that for (3.3) at $(\mathbf{x}^*, \mathbf{u}^*)$ [44].*

By Lemma 1, there exists $r > 0$ such that $\|\mathbf{q} - \mathbf{q}^*\| < r$ implies that the active constraints in (3.19) are the linearization of \bar{G} at $(\mathbf{x}^*, \mathbf{u}^*)$. Thus, (3.19) is equivalent to the following equality-constrained problem:

$$\begin{aligned} \min_{\hat{\mathbf{q}}} \quad & \frac{1}{2}(\hat{\mathbf{q}} - \mathbf{q}_r)^\top P(\hat{\mathbf{q}} - \mathbf{q}_r) \\ \text{s.t.} \quad & \bar{G}(\mathbf{q}) + d\bar{G}_q(\hat{\mathbf{q}} - \mathbf{q}) = 0. \end{aligned} \quad (3.20)$$

As \bar{G} is C^2 , the function $[\bar{G}^\top \ I]^\top$ is Lipschitz with some Lipschitz constant $L > 0$. The following theorem shows that the solutions to (3.20) are contracting towards \mathbf{q}^* .

Theorem 2. *Under Assumption 1, there exists $\hat{r} > 0$ such that $\|\mathbf{q}_s - \mathbf{q}^*\| < \hat{r}$ implies that the solution to (3.20) satisfies:*

$$\|\hat{\mathbf{q}} - \mathbf{q}^*\| \leq k \|\mathbf{q} - \mathbf{q}^*\|, \quad k, kL \in (0, 1). \quad (3.21)$$

Proof. This proof is adapted from the Proof of Theorem 5 of [15]. In [15], the scenario where w_q is a vector is investigated, whereas in this work, we consider a generalization where w_q is a matrix. For Problem (3.18), introduce Lagrange multipliers $\lambda \in \mathbb{R}^{(n+m)T+\bar{p}}$ where \bar{p} is the total number of active inequality constraints and write the Lagrangian, take its derivative, set to zero to get

$$P(\mathbf{q}^* - \mathbf{q}_r) = -d\bar{G}_{\mathbf{q}^*}^\top \lambda^*, \quad (3.22)$$

where λ^* are the Lagrange multipliers at \mathbf{q}^* . Let $w_q = d\bar{G}_{\mathbf{q}^*}^\top$.

For Problem (3.20), introduce Lagrange multipliers $\hat{\lambda} \in \mathbb{R}^{(n+m)T+\bar{p}}$, write the Lagrangian, take its derivative and set to zero to get

$$\begin{bmatrix} P & w_q \\ w_q^\top & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{q}} \\ \hat{\lambda} \end{bmatrix} = \begin{bmatrix} P\mathbf{q}_r \\ w_q^\top \mathbf{q} - \bar{G}(\mathbf{q}) \end{bmatrix}.$$

Inverting the matrix on the left hand side, the solution to this system can be shown to be

$$\begin{aligned} \hat{\mathbf{q}} = & P^{-1} (I - w_q(w_q^\top P^{-1} w_q)^{-1} w_q^\top P^{-1}) P\mathbf{q}_r \\ & + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} (w_q^\top \mathbf{q} - \bar{G}(\mathbf{q})) =: f(\mathbf{q}). \end{aligned} \quad (3.23)$$

By Assumption 1, there exists $\hat{r} > 0$ such that for all \mathbf{q} such that $\|\mathbf{q} - \mathbf{q}^*\| < \hat{r}$, $w_q \neq 0$. As P is symmetric positive definite, this implies that Equation (3.23) is well-defined so there exists a unique solution $\hat{\mathbf{q}}$.

By the multivariate mean value theorem, for each i there exists $\mathbf{q}^i = t_i \mathbf{q} + (1 - t_i) \mathbf{q}^*$ for some $t_i \in [0, 1]$ such that $\bar{G}_i(\mathbf{q}) = \left. \frac{d\bar{G}_i}{d\mathbf{q}} \right|_{\mathbf{q}^i} (\mathbf{q} - \mathbf{q}^*)$. Define

$$\bar{w}_q^\top := \left[\left. \frac{d\bar{G}_1}{d\mathbf{q}} \right|_{\mathbf{q}^1} \quad \cdots \quad \left. \frac{d\bar{G}_n}{d\mathbf{q}} \right|_{\mathbf{q}^n} \right]^\top,$$

then we have

$$\bar{G}(\mathbf{q}) = \bar{w}_q^\top (\mathbf{q} - \mathbf{q}^*). \quad (3.24)$$

Subtracting \mathbf{q}^* from $\hat{\mathbf{q}}$ and substituting in Equation (3.22) and (3.24), we get the following expression.

$$\begin{aligned} \hat{\mathbf{q}} - \mathbf{q}^* &= P^{-1} (I - w_q (w_q^\top P^{-1} w_q)^{-1} w_q^\top P^{-1}) P \mathbf{q}_r \\ &\quad + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} (w_q^\top \mathbf{q} - \bar{G}(\mathbf{q})) - \mathbf{q}^* \\ &= \mathbf{q}_r - \mathbf{q}^* - P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} w_q^\top \mathbf{q}_r \\ &\quad + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} (w_q^\top \mathbf{q} - \bar{G}(\mathbf{q})). \end{aligned}$$

Substitute Equation (3.24) into the above expression to get:

$$\begin{aligned} \hat{\mathbf{q}} - \mathbf{q}^* &= \mathbf{q}_r - \mathbf{q}^* - P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} w_q^\top \mathbf{q}_r \\ &\quad + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} (w_q^\top \mathbf{q} - \bar{w}_q^\top (\mathbf{q} - \mathbf{q}^*)) \\ &= \mathbf{q}_r - \mathbf{q}^* + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} w_q^\top (\mathbf{q} - \mathbf{q}_r) \\ &\quad + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} \bar{w}_q^\top (\mathbf{q}^* - \mathbf{q}). \end{aligned}$$

Writing $\mathbf{q}_r - \mathbf{q}^* = \mathbf{q}_r - \mathbf{q} + \mathbf{q} - \mathbf{q}^*$, we get:

$$\begin{aligned} \hat{\mathbf{q}} - \mathbf{q}^* &= (\mathbf{q} - \mathbf{q}^*) + (\mathbf{q}_r - \mathbf{q}) + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} w_q^\top (\mathbf{q} - \mathbf{q}_r) \\ &\quad + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} \bar{w}_q^\top (\mathbf{q}^* - \mathbf{q}). \end{aligned}$$

Writing $\mathbf{q} - \mathbf{q}_r = \mathbf{q} - \mathbf{q}^* + \mathbf{q}^* - \mathbf{q}_r$, we get:

$$\begin{aligned} \hat{\mathbf{q}} - \mathbf{q}^* &= (\mathbf{q} - \mathbf{q}^*) + (\mathbf{q}_r - \mathbf{q}) + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} w_q^\top [(\mathbf{q} - \mathbf{q}^*) + (\mathbf{q}^* - \mathbf{q}_r)] \\ &\quad + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} \bar{w}_q^\top (\mathbf{q}^* - \mathbf{q}) \\ &= (\mathbf{q} - \mathbf{q}^*) + (\mathbf{q}_r - \mathbf{q}) + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} w_q^\top (\mathbf{q} - \mathbf{q}^*) \\ &\quad + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} w_q^\top (\mathbf{q}^* - \mathbf{q}_r) + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} \bar{w}_q^\top (\mathbf{q}^* - \mathbf{q}) \\ &= (\mathbf{q} - \mathbf{q}^*) + (\mathbf{q}_r - \mathbf{q}) + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} (w_q^\top - \bar{w}_q^\top) (\mathbf{q} - \mathbf{q}^*) \\ &\quad + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} w_q^\top (\mathbf{q}^* - \mathbf{q}_r) \\ &= (\mathbf{q}_r - \mathbf{q}) + [I + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} (w_q^\top - \bar{w}_q^\top)] (\mathbf{q} - \mathbf{q}^*) \\ &\quad + P^{-1} w_q (w_q^\top P^{-1} w_q)^{-1} w_q^\top (\mathbf{q}^* - \mathbf{q}_r). \end{aligned}$$

Now, look at the last term of the expression of $\hat{\mathbf{q}} - \mathbf{q}^*$ and substitute Equation (3.22) to get the following.

$$\begin{aligned}
& P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}w_q^\top(\mathbf{q}^* - \mathbf{q}_r) \\
&= P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}w_q^\top P^{-1}w_{q^*}(-\lambda^*) \\
&= P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}w_q^\top P^{-1}(w_{q^*} - w_q + w_q)(-\lambda^*) \\
&= P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}w_q^\top P^{-1}(w_{q^*} - w_q)(-\lambda^*) + P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}w_q^\top P^{-1}w_q(-\lambda^*). \\
&= P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}w_q^\top P^{-1}(w_{q^*} - w_q)(-\lambda^*) + P^{-1}w_q(-\lambda^*) \\
&= P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}w_q^\top P^{-1}(w_{q^*} - w_q)(-\lambda^*) + P^{-1}(w_q - w_{q^*} + w_{q^*})(-\lambda^*) \\
&= P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}w_q^\top P^{-1}(w_{q^*} - w_q)(-\lambda^*) + P^{-1}w_{q^*}(-\lambda^*) + P^{-1}(w_q - w_{q^*})(-\lambda^*).
\end{aligned}$$

Now, if we substitute Equation (3.22) into the above expression, we will get:

$$\begin{aligned}
& P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}w_q^\top(\mathbf{q}^* - \mathbf{q}_r) \\
&= P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}w_q^\top P^{-1}(w_{q^*} - w_q)(-\lambda^*) + (\mathbf{q}^* - \mathbf{q}_r) + P^{-1}(w_q - w_{q^*})(-\lambda^*).
\end{aligned}$$

Substituting this expression back into the expression of $\hat{\mathbf{q}} - \mathbf{q}^*$, we get:

$$\begin{aligned}
\hat{\mathbf{q}} - \mathbf{q}^* &= (\mathbf{q}_r - \mathbf{q}) + [I + P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}(w_q^\top - \bar{w}_q^\top)](\mathbf{q} - \mathbf{q}^*) \\
&\quad + P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}w_q^\top P^{-1}(w_{q^*} - w_q)(-\lambda^*) \\
&\quad + (\mathbf{q}^* - \mathbf{q}_r) + P^{-1}(w_q - w_{q^*})(-\lambda^*) \\
&= P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}(w_q - \bar{w}_q)^\top(\mathbf{q} - \mathbf{q}^*) \\
&\quad + P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1}w_q^\top P^{-1}(w_{q^*} - w_q)(-\lambda^*) + P^{-1}(w_q - w_{q^*})(-\lambda^*).
\end{aligned}$$

Now we multiply both sides of the expression by w_q^\top and notice that

$$w_q^\top P^{-1}w_q(w_q^\top P^{-1}w_q)^{-1} = (w_q^\top P^{-1}w_q)(w_q^\top P^{-1}w_q)^{-1} = I,$$

which simplifies the expression to

$$\begin{aligned}
& w_q^\top(\hat{\mathbf{q}} - \mathbf{q}^*) \\
&= (w_q - \bar{w}_q)^\top(\mathbf{q} - \mathbf{q}^*) + w_q^\top P^{-1}(w_{q^*} - w_q)(-\lambda^*) + w_q^\top P^{-1}(w_q - w_{q^*})(-\lambda^*) \quad (3.25) \\
&= (w_q - \bar{w}_q)^\top(\mathbf{q} - \mathbf{q}^*).
\end{aligned}$$

As $\|d\bar{G}_q\|$ is continuous over $\bar{B}_r(\mathbf{q}^*)$ compact and w_{q^*} is nonzero so $\|w_{q^*}^\top\|_P = \|d\bar{G}_{q^*}\|_P > 0$, by shrinking r if necessary, there exist $c_m > 0$ and $c_M > 0$ such that

$$0 < c_m \leq \|d\bar{G}_q\|_P \leq c_M \quad (3.26)$$

for all $\mathbf{q} \in \overline{B_{\hat{r}}}(\mathbf{q}^*)$.

Shrink r if necessary and choose $d > 0$ such that

$$k := \frac{d}{c_m} < \min\{L^{-1}, 1\}. \quad (3.27)$$

As $\|d\bar{G}_q\|$ is continuous over $\overline{B_r}(\mathbf{q}^*)$ compact, it is uniformly continuous, so by shrinking r if necessary, there exists $\alpha > 0$ such that for $p, q \in \overline{B_r}(\mathbf{q}^*)$, $\|\mathbf{q} - \mathbf{p}\| < \alpha$ implies that

$$\|d\bar{G}_q - d\bar{G}_p\|_P < d. \quad (3.28)$$

Now expressing $\hat{\mathbf{q}} - \mathbf{q}^*$ as Equation (3.25), and bounding the norm of both sides using (3.26) and (3.28), we get that

$$\begin{aligned} \|w_q^\top(\hat{\mathbf{q}} - \mathbf{q}^*)\|_P &\geq c_m \|\hat{\mathbf{q}} - \mathbf{q}^*\|_P. \\ \|(w_q^\top - \bar{w}_q)(\mathbf{q} - \mathbf{q}^*)\|_P &\leq d \|\mathbf{q} - \mathbf{q}^*\|_P. \end{aligned}$$

Putting it all together, we get that

$$\|\hat{\mathbf{q}} - \mathbf{q}^*\|_P \leq \frac{d}{c_m} \|\mathbf{q} - \mathbf{q}^*\|_P.$$

Hence,

$$\|\hat{\mathbf{q}} - \mathbf{q}^*\| \leq k \|\mathbf{q} - \mathbf{q}^*\|.$$

□

Next, write $\hat{\mathbf{q}} = [\hat{\mathbf{x}}^\top \ \hat{\mathbf{u}}^\top]^\top$, we will show that the contraction of $\hat{\mathbf{q}}$ towards \mathbf{q}^* obtained from Theorem 2 also leads to a contraction of $\hat{\mathbf{u}}$ towards \mathbf{u}^* alone.

Corollary 3. *Under the conditions of Theorem 2, there exists $\hat{r} > 0$ such that $\|\mathbf{u} - \mathbf{u}^*\| < \hat{r}$ and $\mathbf{x} = G(x^0, \mathbf{u})$ implies that the solution to (3.20) satisfies:*

$$\|\hat{\mathbf{u}} - \mathbf{u}^*\| \leq k \|\mathbf{u} - \mathbf{u}^*\|, \quad k \in (0, 1). \quad (3.29)$$

Proof. Define $F := [\bar{G}^\top \ I]^\top$ and recall F is Lipschitz with Lipschitz constant L . Applying Theorem 2, we compute

$$\begin{aligned} \|\hat{\mathbf{u}} - \mathbf{u}^*\| &\leq \left\| \begin{bmatrix} \hat{\mathbf{x}} - \mathbf{x}^* \\ \hat{\mathbf{u}} - \mathbf{u}^* \end{bmatrix} \right\| \leq \bar{k} \left\| \begin{bmatrix} \mathbf{x} - \mathbf{x}^* \\ \mathbf{u} - \mathbf{u}^* \end{bmatrix} \right\| \\ &= \bar{k} \left\| \begin{bmatrix} G(x^0, \mathbf{u}) - G(x^0, \mathbf{u}^*) \\ \mathbf{u} - \mathbf{u}^* \end{bmatrix} \right\| \\ &= \bar{k} \|F(x^0, \mathbf{u}) - F(x^0, \mathbf{u}^*)\| \\ &\leq \bar{k}L \|\mathbf{u} - \mathbf{u}^*\|, \end{aligned}$$

where $k := \bar{k}L < 1$ by Theorem 2. □

Recall that problem (3.13) is the subproblem (3.19) with trust region constraints included. By Lemma 1, for $\mathbf{q}_s \in B_r(\mathbf{q}^*)$, problem (3.13) is equivalent to

$$\begin{aligned} \min_{\mathbf{q}_{s+1}} \quad & \frac{1}{2}(\mathbf{q}_{s+1} - \mathbf{q}_r)^\top P(\mathbf{q}_{s+1} - \mathbf{q}_r) \\ \text{s.t.} \quad & \bar{G}(\mathbf{q}_s) + d\bar{G}_{\mathbf{q}_s}(\mathbf{q}_{s+1} - \mathbf{q}_s) = 0, \\ & \|\mathbf{u}_{s+1} - \mathbf{u}_s\| \leq \Delta. \end{aligned} \tag{3.30}$$

The following theorem provides a local convergence guarantee for Algorithm 1.

Theorem 4. *Suppose $H(x^0, \mathbf{u}) = M\mathbf{u}$ for some constant matrix M . Assume the conditions of Lemma 1 and Corollary 3. There exist $\bar{r} > \hat{r}$ and $\bar{\Delta} > 0$ such that for any $\Delta \in (0, \bar{\Delta})$ and any \mathbf{u}_0 such that $\|\mathbf{u}_0 - \mathbf{u}^*\| \leq \bar{r}$, the sequence of solutions obtained from subproblem (3.13), denoted $\{\mathbf{u}_s\}_{s=0}^\infty$, converges to \mathbf{u}^* .*

Remark 5. This theorem provides a local convergence guarantee for the proposed method in the important special case where H is a linear function of inputs, as in the example provided in Chapter 5.2.1 and 5.2.2. Future work will aim to extend this to more general inequality constraints H .

Proof. By Equation (3.23), $\hat{\mathbf{u}} = f(\mathbf{u})$ is continuous. By Corollary 3, f is a contraction over $B_{\hat{r}}(\mathbf{u}^*)$. Thus, there exists $\tilde{r} > \hat{r}$ such that $\mathbf{u} \in B_{\tilde{r}}(\mathbf{u}^*)$ implies that $f(\mathbf{u}) \in \bar{B}_{\tilde{r}}(\mathbf{u}^*)$. Again, since $f(\mathbf{u})$ is continuous, $\|\hat{\mathbf{u}} - \mathbf{u}\| = \|f(\mathbf{u}) - \mathbf{u}\|$ achieves a minimum over the compact set $\bar{B}_{\tilde{r}}(\mathbf{u}^*) - B_{\tilde{r}}(\mathbf{u}^*)$, say c . By Corollary 3, f is a contraction over $B_{\tilde{r}}(\mathbf{u}^*)$ with a unique fixed point $f(\mathbf{u}^*) = \mathbf{u}^*$, so $c > 0$ since f does not have a fixed point on the set $\bar{B}_{\tilde{r}}(\mathbf{u}^*) - B_{\tilde{r}}(\mathbf{u}^*)$. Let $\bar{\Delta} = c$. Fix any $\Delta \in (0, \bar{\Delta})$.

Since H is linear, the inequality constraints in Problem (3.13) are exact and have no approximations. Consider Problem (3.13) at iteration $(s-1)$, the inputs to the QP include constant vectors \mathbf{x}_{s-1} , \mathbf{u}_{s-1} , and \mathbf{h}_{s-1} ; the solution to the problem $d\mathbf{u}$ can be used to update inputs as $\mathbf{u}_s = \mathbf{u}_{s-1} + d\mathbf{u}$, and \mathbf{u}_s can also be treated as solution to the problem. Any such solution \mathbf{u}_s satisfies (3.9) resulting in $H(x^0, \mathbf{u}_s) \leq 0$, and the set of active inequality constraints then satisfy $\bar{H}(x^0, \mathbf{u}_s) = 0$.

By Algorithm 1 step 3, we compute the exact vehicle trajectory according to system dynamics as $\mathbf{x}_s = G(x^0, \mathbf{u}_s)$. Now recall the definition of \bar{G} in (3.17), and substituting in the values from the above analysis

$$\bar{G}(\mathbf{x}_s, \mathbf{u}_s) = \begin{bmatrix} \mathbf{x}_s - G(x^0, \mathbf{u}_s) \\ \bar{H}(x^0, \mathbf{u}_s) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Therefore, the following holds true

$$\bar{G}(\mathbf{q}_s) = \bar{G}(\mathbf{x}_s, \mathbf{u}_s) = 0.$$

Recall that Problem (3.13) is equivalent to Problem (3.30) based on our assumptions, which implies that any solution \mathbf{q}_s to Problem (3.30) satisfies $\bar{G}(\mathbf{q}_s) = 0$.

Recall that without trust region constraints, $\hat{\mathbf{q}}$ is the solution to subproblem (3.20). Measuring distance using the inner product induced by P , it follows that the solution $\hat{\mathbf{q}}$ to Problem (3.20) with $\mathbf{q} = \mathbf{q}_s$ is the unique closest point on the following hyperplane to \mathbf{q}_r

$$\bar{G}(\mathbf{q}_s) + d\bar{G}_{q_s}(\mathbf{q} - \mathbf{q}_s) = d\bar{G}_{q_s}(\mathbf{q} - \mathbf{q}_s) = 0.$$

Let S denote the intersection of $d\bar{G}_{q_s}^\top(\mathbf{q} - \mathbf{q}_s) = 0$ and $\mathbb{R}^{n^T} \times \bar{B}_\Delta(\mathbf{u}_s)$. The solution \mathbf{q}_{s+1} to (3.30) is the unique closest point in S to \mathbf{q}_r . Since $\mathbf{q}_s \in S$ because $\bar{G}(\mathbf{q}_s) = 0$, it is straight forward to show that the closest point in S to \mathbf{q}_r is also the closest point in S to $\hat{\mathbf{q}}$. This latter point is equal to the intersection of the line segment from \mathbf{q}_s to $\hat{\mathbf{q}}$ and the boundary of $\mathbb{R}^{n^T} \times \bar{B}_\Delta(\mathbf{u}_s)$. A point on this line segment is

$$(1 - \gamma)\mathbf{q}_s + \gamma\hat{\mathbf{q}} = \mathbf{q}(\gamma) = [\mathbf{x}(\gamma)^\top \quad \mathbf{u}(\gamma)^\top]^\top, \quad (3.31)$$

where $\gamma \in (0, 1)$. Then $\mathbf{q}(\gamma)$ intersects the boundary of the $\mathbb{R}^{n^T} \times \bar{B}_\Delta(\mathbf{u}_s)$ if and only if $\|\mathbf{u}(\gamma) - \mathbf{u}_s\| = \Delta$. Substituting in (3.31) and solving for γ yields $\hat{\gamma} = \frac{\Delta}{\|\hat{\mathbf{u}} - \mathbf{u}_s\|}$, so $\mathbf{u}(\hat{\gamma})$ becomes

$$\mathbf{u}(\hat{\gamma}) = (1 - \hat{\gamma})\mathbf{u}_s + \hat{\gamma}\hat{\mathbf{u}}, \quad (3.32)$$

then

$$\mathbf{u}_{s+1} = \begin{cases} \hat{\mathbf{u}} & \text{if } \|\hat{\mathbf{u}} - \mathbf{u}_s\| \leq \Delta \\ \mathbf{u}(\hat{\gamma}) & \text{if } \|\hat{\mathbf{u}} - \mathbf{u}_s\| > \Delta. \end{cases} \quad (3.33)$$

When $\mathbf{u}_{s+1} = \mathbf{u}(\hat{\gamma})$, subtract \mathbf{u}^* from \mathbf{u}_{s+1} to get

$$\mathbf{u}_{s+1} - \mathbf{u}^* = (1 - \hat{\gamma})(\mathbf{u}_s - \mathbf{u}^*) + \hat{\gamma}(\hat{\mathbf{u}} - \mathbf{u}^*).$$

Since $\hat{\mathbf{u}} \in B_{\hat{r}}(\mathbf{u}^*)$, taking the norm and substitute Equation (3.29) in, we get

$$\|\mathbf{u}_{s+1} - \mathbf{u}^*\| \leq (1 - \hat{\gamma} + k\hat{\gamma}) \|\mathbf{u}_s - \mathbf{u}^*\|.$$

As $\hat{\gamma}$ is continuous, and the set $\bar{B}_{\hat{r}}(\mathbf{u}^*) - B_{\hat{r}}(\mathbf{u}^*)$ is compact, $\hat{\gamma}$ attains a minimum $\bar{\gamma}$ over the set. Then $\bar{\gamma} < 1$ by the choice of Δ , which implies that

$$1 - \hat{\gamma} + k\hat{\gamma} \leq 1 - \bar{\gamma} + k\bar{\gamma} < 1,$$

so we have a contraction for $\mathbf{u} \in \overline{B_{\hat{r}}(\mathbf{u}^*)} - B_{\hat{r}}(\mathbf{u}^*)$. Thus, if \mathbf{u}_0 is initially in the set $B_{\hat{r}}(\mathbf{u}^*)$, \mathbf{u}_s will enter $B_{\hat{r}}(\mathbf{u}^*)$ in a finite number of iterations.

For any $\mathbf{u}_s \in B_{\hat{r}}(\mathbf{u}^*)$, by (3.33), \mathbf{u}_{s+1} is equal to either $\hat{\mathbf{u}}$ or $\mathbf{u}(\hat{\gamma})$, so

$$\|\mathbf{u}_{s+1} - \mathbf{u}^*\| \leq \hat{k} \|\mathbf{u}_s - \mathbf{u}^*\|,$$

where \hat{k} is either k or $1 - \hat{\gamma} + k\hat{\gamma}$ and is less than 1. Thus, $\|\mathbf{u}_s - \mathbf{u}^*\|$ is monotonically decreasing in s and bounded below by 0, so $\|\mathbf{u}_s - \mathbf{u}^*\|$ converges, say to \hat{c} .

Assume towards a contradiction that $\hat{c} > 0$. Then there exists $\tilde{\mathbf{u}}$ such that $\|\tilde{\mathbf{u}} - \mathbf{u}^*\| = \hat{c}$ and a subsequence of $\{\mathbf{u}_s\}_{s=0}^{\infty}$ converges to $\tilde{\mathbf{u}}$. Let $\tilde{\mathbf{u}}_+$ be the solution of (3.30) with $\tilde{\mathbf{u}}$ in place of \mathbf{u}_s . Then by the same argument as above, $\|\tilde{\mathbf{u}}_+ - \mathbf{u}^*\| \leq \hat{k} \|\tilde{\mathbf{u}} - \mathbf{u}^*\|$ where $\hat{k} < 1$. As the solution of (3.30), which is given by (3.33), is piecewise continuous, and since a subsequence of $\{\mathbf{u}_s\}_{s=0}^{\infty}$ converges to $\tilde{\mathbf{u}}$, there exists \mathbf{u}_s such that

$$\begin{cases} \|\mathbf{u}_s - \tilde{\mathbf{u}}\| < \epsilon_1 \\ \|\mathbf{u}_{s+1} - \mathbf{u}^*\| \leq (\hat{k} + \epsilon_2) \|\mathbf{u}_s - \mathbf{u}^*\| \end{cases}$$

for some $\epsilon_1, \epsilon_2 > 0$ such that

$$\begin{cases} \hat{k} + \epsilon_2 < 1 \\ (\hat{k} + \epsilon_2)(\hat{c} + \epsilon_1) < \hat{c}. \end{cases}$$

Thus we have

$$\|\mathbf{u}_s - \mathbf{u}^*\| < \hat{c} + \epsilon_1,$$

hence,

$$\|\mathbf{u}_{s+1} - \mathbf{u}^*\| < (\hat{k} + \epsilon_2)(\hat{c} + \epsilon_1) < \hat{c},$$

which is a contradiction. So, $\hat{c} = 0$, which implies that $\{\mathbf{u}_s\}_{s=0}^{\infty}$ converges to \mathbf{u}^* .

□

3.4 Adaptation to a Tracking Controller

The vehicle motion planner that we design in this section can be easily modified to become a tracking controller. Looking at the optimization problem in (3.3), the output of our vehicle motion planner includes the required input trajectory \mathbf{u} and the corresponding state trajectory \mathbf{x} . Therefore, \mathbf{u} can be used as a sequence of control commands for a

vehicle. However, uncertainties in real-life environments and system modeling errors can cause the vehicle to produce a trajectory different from the desired trajectory \mathbf{x} . To handle this, we could take real-time measurements of vehicle states as feedback and resolve (3.3) with new initial states of the vehicle. Additionally, we can adopt a Model Predictive Control (MPC) framework, where we solve (3.3) for a long horizon, execute the inputs in the first few time steps, take measurements, and resolve (3.3) for a shifted horizon.

Chapter 4

Driving Style Imitating Problem

In this chapter, we address the second layer of the planning phase of a personalized AV control problem. As presented in Chapter 3, this subproblem is also the UL of a BP problem. A common approach to solving BP problems is the use of gradient-based methods, where the "gradient" refers to the derivatives of the UL objective with respect to the UL optimization variables. In this driving style imitating problem, the UL objective is generally not differentiable with respect to the UL optimization variables given the nonconvex LL problem. Therefore, we refer to these derivatives as subgradients. Mathematically, recall the driving style imitating problem in (2.7) with the following form of objective,

$$J_U(\mathbf{y}_d, \mathbf{y}(\mathbf{u}_*(w))),$$

and its subgradient with respect to w is

$$\frac{dJ_U}{dw},$$

which is dependent on the derivatives of the LL solution (that is, solution to the motion planning problem) with respect to its input parameters,

$$\frac{dJ_U}{dw} = \frac{\partial J_U}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}_*} \frac{\partial \mathbf{u}_*}{\partial w} = \frac{\partial J_U}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}_*} \frac{dF}{dw},$$

where F is the LL solution mapping. Given that F may be nonsmooth, the derivative term $\frac{dF}{dw}$ is a subgradient defined as

$$\frac{dF}{dw} \in \partial F(w),$$

where $\partial F(w)$ is the subdifferential of the function F with respect to w .

To distinguish between the two subgradients, we refer to $\frac{dJ_U}{dw}$ as the **UL** subgradient and to $\frac{dF}{dw}$ as the **LL** subgradient. We show how we perform a sensitivity analysis of the **LL** problem to compute the **LL** subgradient, how the **UL** subgradient is calculated using the **LL** subgradient and how a gradient-based algorithm is constructed to solve the driving style imitating problem.

4.1 Sensitivity Analysis

In this section, we show how the **LL** subgradient (i.e. the derivatives of the solution to the **LL** problem with respect to its weights) is calculated using a sensitivity analysis of the problem. We first derive a matrix equality by solving which the derivatives can be obtained. As some entries in the matrix equality contain second-order **TS**, we then derive expressions for them. Finally, we comment on the existence and uniqueness of solutions to the matrix equality.

4.1.1 Derivation of Expressions

To calculate **LL** subgradients, a sensitivity analysis of an optimization problem can be performed by differentiating the **Karush–Kuhn–Tucker (KKT)** conditions, constructing a matrix equality and solving for a solution. A similar but less rigorous procedure has been used in existing works [1]. We elaborate on this approach when **TS** are used in place of first-order partial derivatives of vehicle dynamics.

Consider the **LL** motion planning problem with input-to-state map, as given in (3.3), and write its Lagrangian

$$L = \frac{1}{2}(\mathbf{x} - \mathbf{x}_r)^T P_1(w_1)(\mathbf{x} - \mathbf{x}_r) + \frac{1}{2}\mathbf{u}^T P_2(w_2)\mathbf{u} + \lambda^T(G(\mathbf{u}) - \mathbf{x}) + \nu^T H(\mathbf{u}) - \mathbf{1}_{\nu \geq 0},$$

where $\lambda \in \mathbb{R}^{nT}$ and $\nu \in \mathbb{R}^p$ are Lagrange multiplier vectors of nT equality and p inequality constraints. The fixed initial conditions x^0 have been dropped from the functions G and H for simplicity. We note that the Lagrangian is a function of $\mathbf{x}, \mathbf{u}, w_1, w_2, \lambda, \nu$. At optimality, we have the following:

$$\nabla L = 0 =: f(\mathbf{x}, \mathbf{u}, w_1, w_2, \lambda, \nu),$$

which evaluates to

$$\begin{bmatrix} P_1(\mathbf{x} - \mathbf{x}_r) - \lambda \\ P_2\mathbf{u} + \frac{dG^T}{d\mathbf{u}}\lambda + \frac{dH^T}{d\mathbf{u}}\nu \\ G(\mathbf{u}) - \mathbf{x} \\ \text{diag}(\nu)H(\mathbf{u}) \end{bmatrix} = \begin{bmatrix} P_1(\mathbf{x} - \mathbf{x}_r) - \lambda \\ P_2\mathbf{u} + \frac{dG^T}{d\mathbf{u}}\lambda + \frac{dH^T}{d\mathbf{u}}\nu \\ G(\mathbf{u}) - \mathbf{x} \\ \text{diag}(H(\mathbf{u}))\nu \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Using the Implicit Function Theorem, taking the differentials of the above equality with respect to w_1 and applying the chain rule, we get [30].

$$\frac{df}{dw_1} = \frac{\partial f}{\partial w_1} + \frac{\partial f}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial w_1} + \frac{\partial f}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial w_1} + \frac{\partial f}{\partial \lambda} \frac{\partial \lambda}{\partial w_1} + \frac{\partial f}{\partial \nu} \frac{\partial \nu}{\partial w_1} = 0.$$

Taking the differentials with respect to w_2 generates the exact same expression, with w_2 in place of w_1 . Combining the expressions of both differentials and writing them compactly in matrix form, we get

$$\underbrace{\begin{bmatrix} \frac{\partial f}{\partial \mathbf{x}} & \frac{\partial f}{\partial \mathbf{u}} & \frac{\partial f}{\partial \lambda} & \frac{\partial f}{\partial \nu} \end{bmatrix}}_A \underbrace{\begin{bmatrix} \frac{\partial \mathbf{x}}{\partial w} \\ \frac{\partial \mathbf{u}}{\partial w} \\ \frac{\partial \lambda}{\partial w} \\ \frac{\partial \nu}{\partial w} \end{bmatrix}}_X = \underbrace{\begin{bmatrix} -\frac{\partial f}{\partial w_1} & -\frac{\partial f}{\partial w_2} \end{bmatrix}}_B.$$

Solving for the expressions of each partial derivative term in matrix A and B , we get

$$\begin{bmatrix} P_1 & 0 & -I & 0 \\ 0 & P_2 + \frac{d}{d\mathbf{u}}(dG_u^T\lambda) + \frac{d}{d\mathbf{u}}(dH_u^T\nu) & dG_u^T & dH_u^T \\ -I & dG_u & 0 & 0 \\ 0 & D(\nu)dH_u & 0 & D(H(u)) \end{bmatrix} \begin{bmatrix} \frac{\partial \mathbf{x}}{\partial w} \\ \frac{\partial \mathbf{u}}{\partial w} \\ \frac{\partial \lambda}{\partial w} \\ \frac{\partial \nu}{\partial w} \end{bmatrix} = \begin{bmatrix} \frac{\partial P_1(\mathbf{x}_r - \mathbf{x})}{\partial w_1} & 0 \\ 0 & -\frac{\partial P_2\mathbf{u}}{\partial w_2} \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad (4.1)$$

where first-order TS $\frac{dG}{d\mathbf{u}}$ and $\frac{dH}{d\mathbf{u}}$ are written as dG_u and dH_u for brevity. By solving the matrix equality of $AX = B$, we obtain X , whose second entry is

$$\frac{\partial \mathbf{u}}{\partial w} = \frac{dF}{dw},$$

which is the LL subgradient that we want to compute. See Appendix A.1 for an analysis of the uniqueness and existence of a solution to this matrix equality.

The matrix equality in (4.1) is different from the same equality that one would obtain without the input-to-state map and the use of first- and second-order trajectory sensitivities. The derivation of the other matrix equality where vehicle dynamics are linearized using Taylor's first-order approximation can be found in Appendix A.2 for the purpose of demonstration.

4.1.2 Second-Order Trajectory Sensitivities Computation

The (2,2)-entry of the matrix equality (4.1) contains two second-order TS terms:

$$\begin{aligned}\frac{d}{d\mathbf{u}}(dG_u^T \lambda) &= \lambda^T \frac{d^2 G}{d\mathbf{u}^2}, \\ \frac{d}{d\mathbf{u}}(dH_u^T \nu) &= \nu^T \frac{d^2 H}{d\mathbf{u}^2}.\end{aligned}$$

For clarity, from this point on, I will use Einstein's summation notation to work with second-order TS. Begin with TS of G , recall that vehicle state is denoted by $x \in \mathbb{R}^n$. Then for a specific state with index $i \in \{1, \dots, n\}$, we can write its dynamics as

$$(x^i)^+ = g^i(x, u).$$

Note that we are now using the superscript to denote an entry of the state vector and vector function. This conflicts with our early use of the superscript as time step, but is required for the new notation. This abuse of notation is only valid in this section for the purpose of calculating second-order TS.

Differentiate the above system dynamics to obtain a formula for first-order trajectory sensitivities:

$$(x_j^i)^+ = g_\alpha^i x_j^\alpha + g_j^i.$$

The subscript j represents a partial derivative with respect to u_j , where $j \in \{1, \dots, mT\}$ is the index of an entry in \mathbf{u} . The superscripts and subscripts $\alpha, \beta \in \{1, \dots, n\}$ represent derivatives with respect to the states.

Differentiate again to obtain an expression to compute second-order TS:

$$(x_{jk}^i)^+ = g_{\alpha\beta}^i x_j^\alpha x_k^\beta + g_{\alpha k}^i x_j^\alpha + g_\alpha^i x_{jk}^\alpha + g_{j\beta}^i x_k^\beta + g_{jk}^i. \quad (4.2)$$

The subscripts j and k represent partial derivatives with respect to u_j and u_k , respectively, where $j, k \in \{1, \dots, mT\}$. The superscripts α and β are again used for state derivatives. Note that the terms $g_{\alpha k}^i x_j^\alpha$ and $g_{j\beta}^i x_k^\beta$ are permutations of each other. In addition, we note that the terms x_j^α and x_k^β are entries of first-order TS, and the term g_α^i is the same as $\frac{\partial g^i}{\partial x_\alpha}$. These terms have already been computed in Chapter 3.1.2.

Similarly, the second-order TS of H can be computed using the expression

$$(h_{jk}^i)^+ = h_{\alpha\beta}^i x_j^\alpha x_k^\beta + h_{\alpha k}^i x_j^\alpha + h_\alpha^i x_{jk}^\alpha + h_{j\beta}^i x_k^\beta + h_{jk}^i. \quad (4.3)$$

4.2 Upper-Level Subgradient Computation

Recall that the expression of **UL** subgradient is given by

$$\frac{dJ_U}{dw} = \frac{\partial J_U}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}_*} \frac{dF}{dw}. \quad (4.4)$$

In Chapter 4.1, we discuss how the **LL** subgradient $\frac{dF}{dw}$ is calculated. We also note that J_U is an explicit function of \mathbf{y} , hence the term $\frac{\partial J_U}{\partial \mathbf{y}}$ is easy to compute. We now explore how the remaining term can be computed efficiently.

In (2.3), it is given that $y^k = Cx^k \forall k \in \{0, 1, \dots, T-1\}$. Defining a new matrix \bar{C} such that

$$\bar{C} = \begin{bmatrix} C & 0 & 0 & \dots & 0 \\ 0 & C & 0 & \dots & 0 \\ 0 & 0 & C & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & C \end{bmatrix} \in \mathbb{R}^{2T \times nT}.$$

Then we can write

$$\mathbf{y} = \bar{C}\mathbf{x},$$

from which we note that

$$\frac{\partial \mathbf{y}}{\partial \mathbf{u}_*} = \frac{\partial \bar{C}\mathbf{x}}{\partial \mathbf{u}_*} = \bar{C} \frac{\partial \mathbf{x}}{\partial \mathbf{u}_*},$$

where $\frac{\partial \mathbf{x}}{\partial \mathbf{u}_*}$ is the first-order **TS** and can be computed as described in Chapter 3.1.2.

The **UL** subgradients calculated in this section are verified via a comparison to the finite-difference approximation results.

4.3 Algorithm Design

To solve the **BP** problem, we adopt a gradient-based algorithm in which the **UL** decision variables, w , are updated along the negative direction of the **UL** subgradient, $\frac{dJ_U}{dw}$. Due to the nonsmoothness of the objective function, the algorithm is a type of subgradient method. Subgradient methods have been discussed in a course note by Stephen Boyd and are not descent methods as the objective value may increase over iterations given the nonsmoothness of the objective function [6].

Let $\alpha_s \in \mathbb{R}$ be the step length in the s th iteration. Then the weights are updated as

$$\bar{w}_{s+1} = w_s - \bar{\alpha}_s \frac{dJ_U}{dw},$$

where $\bar{\alpha}_s \in \mathbb{R}$ such that $\|\bar{\alpha}_s \frac{dJ_U}{dw}\| = \alpha_s$. The magnitude of the step length α_s is chosen first as a constant value and is then set to decrease over iterations as the change in objective value becomes small. This is a combination of fixed step length and vanishing step length subgradient update methods. We choose this step-length update scheme by running the algorithm with some test examples and observing the convergence trend. In addition to this, there are many other update methods, including the vanishing step size method, the variable step size method through a line search procedure, etc. [6]

Additionally, we note that the weight coefficients must satisfy the element-wise inequality constraint:

$$w > 0.$$

To ensure that this constraint is met, we introduce a projection step after the **UL** subgradient update. Let S be a nonempty subset of \mathbb{R}^n , let $a \in \mathbb{R}^n$ and let $b \in \mathbb{R}^n$. Then b is a projection of a onto S if

$$b \in \operatorname{argmin}_{c \in S} \|c - a\|^2.$$

In this scenario, let $S = \mathbb{R}_{++}^n$ be a closed, nonempty convex set; thus the projection problem has a unique solution of

$$\mathbb{P}_S(\bar{w}_{s+1}) = w_{s+1} = \begin{cases} \bar{w}_{s+1}^i, & \bar{w}_{s+1}^i \geq \epsilon_w, \\ \epsilon_w, & \bar{w}_{s+1}^i < \epsilon_w, \end{cases}$$

where ϵ_w is a small positive constant and $i \in \{1, \dots, l\}$ is the index of an entry in vector w . To implement this, we simply set $w^i \in \{w^i \mid w^i \leq \epsilon_w\}$ to ϵ_w .

This projected subgradient update algorithm is summarized below. In Algorithm 2, we perform the algorithm initialization on lines 1 and 2. Then, we solve the **LL** vehicle motion planning problem using Algorithm 1 on line 4. On lines 5, 6 and 7, we compute the **UL** subgradient. On lines 8, 9 and 10, we determine whether the step length should be decreased. We then perform a subgradient update on line 11 and project it to the feasible weight set on line 12. Convergence is determined by the magnitude of the step length of subgradient update.

Remark 6. In Chapter 2.3.4, we introduced the concept of Clarke Jacobians, which is a more accurate generalization of differentials to be used in this context. Clarke Jacobians

Algorithm 2 Projected Subgradient Update

- 1: Choose initial weight vector w_s and step length α_s ; set $s \leftarrow 0$
 - 2: Initialize J_{s-1} to a large value
 - 3: **while** Convergence test not satisfied **do**
 - 4: Solve the **LL** problem with parameter w_s using **SQP-TS algorithm** to get $(\mathbf{x}_*, \mathbf{u}_*)$
 - 5: Compute first- and second- order **TS** at $(\mathbf{x}_*, \mathbf{u}_*)$ using (3.4), (3.5), (4.2) and (4.3)
 - 6: Compute **UL** subgradient $\frac{dJ_U}{dw}$ at $(w_s, \mathbf{x}_*, \mathbf{u}_*)$ using (4.4)
 - 7: Evaluate $J_s = J_U(w_s, \mathbf{x}_*, \mathbf{u}_*)$; compute $\Delta J = J_s - J_{s-1}$; set $J_{s-1} \leftarrow J_s$
 - 8: **if** ΔJ smaller than some tolerance **then**
 - 9: Decrease α_s
 - 10: **end if**
 - 11: Compute $\bar{\alpha}_s$; set $\bar{w}_{s+1} = w_s - \bar{\alpha}_s \frac{dJ_U}{dw}$ such that $\|\bar{\alpha}_s \frac{dJ_U}{dw}\| = \alpha_s$
 - 12: Set $w_{s+1} = \mathbb{P}_{\mathbb{R}_+}(\bar{w}_{s+1})$; set $s \leftarrow s + 1$
 - 13: **end while**
-

are the same as subdifferentials at differentiable points and only differ at non-differentiable points. In implementation, we observe that applying the subdifferentials directly without identifying points of nonsmoothness already has good performance. Therefore, we consider the subgradient as update direction in Algorithm 2 as opposed to element of Clarke Jacobians. Further theoretical analysis is under investigation to backup this simplification.

Chapter 5

Simulation

In this chapter, we present the simulation results of the vehicle motion planning problem using [SQP-TS](#), and the simulation results of the driving style imitating problem using projected subgradient descent. For the former problem, we use a single-track bicycle model of a vehicle and set the reference paths to include smooth, nonsmooth, obstacle-free and obstacle-present paths. We compare the performance of our proposed algorithm with the traditional [SQP](#) method. For the latter problem, we use a dynamic single-track model of vehicle and use demonstrations with consistent and inconsistent driving styles in environments with and without obstacles.

5.1 Motion Planning Problem: Setup

In this section, we demonstrate how the proposed approach is applied to the problem of trajectory optimization of an autonomous vehicle, which is also the vehicle motion planning part of the [BP](#) problem. The proposed [SQP-TS](#) algorithm is implemented in Python. Each quadratic subproblem is solved using the commercially available Gurobi solver [\[20\]](#).

5.1.1 Single-Track Bicycle Model

We consider the application of generating an optimal vehicle trajectory given some reference paths. To model vehicle dynamics, a single-track bicycle model is used, which requires the following set of state variables and input.

$$x = [v_x \quad v_y \quad \theta \quad \dot{\theta} \quad P_x \quad P_y]^\top, \quad u = \delta,$$

where $x \in \mathbb{R}^6$ is the state vector, $u \in \mathbb{R}$ is the input, v_x and v_y are longitudinal and lateral velocities of the vehicle in the global frame, θ and $\dot{\theta}$ are the heading angle and angular velocity, P_x and P_y are longitudinal and lateral displacements in the global frame, and δ is the steering angle.

Following [10], some assumptions and simplifications are made to simplify the dynamic vehicle model, and the following CT model is obtained.

$$\dot{x} = \begin{bmatrix} v_y \dot{\theta} + \frac{s_f C_{xf} + s_r C_{xr}}{m} - \frac{C_{\alpha_f}(\frac{\dot{y} - b\dot{\theta}}{v_x} - \delta)\delta}{m} \\ -v_x \dot{\theta} + \frac{C_{\alpha_f}(\frac{v_y + a\dot{\theta}}{v_x} - \delta)}{m} + \frac{C_{\alpha_r}(\frac{v_y - b\dot{\theta}}{v_x})}{m} \\ \dot{\theta} \\ \frac{1}{I_z}(aC_{\alpha_f}(\frac{v_y + a\dot{\theta}}{v_x} - \delta) - bC_{\alpha_r}(\frac{v_y - b\dot{\theta}}{v_x})) \\ v_x \cos \theta - v_y \sin \theta \\ v_x \sin \theta + v_y \cos \theta \end{bmatrix},$$

where the new variables $a, b, s_f, s_r, C_{\alpha_f}, C_{\alpha_r}, C_{xf}, C_{xr}, m, I_z$ are vehicle-related parameters, and their definitions can be found in Appendix B.2 and in [10]. For simplicity, we define some constant terms as follows.

$$C_1 = \frac{s_f C_{xf} + s_r C_{xr}}{m}, \quad C_2 = \frac{C_{\alpha_f}}{m}, \quad C_3 = \frac{C_{\alpha_r}}{m}, \quad C_4 = \frac{bC_{\alpha_r}}{I_z}, \quad C_5 = \frac{aC_{\alpha_f}}{I_z}$$

Choosing a fixed sample time T_s , this nonlinear model can be discretized into the form

$$x^{k+1} = \begin{bmatrix} x_1^k + T_s(x_2^k x_4^k + C_1) & -T_s(C_2 \frac{x_2^k + ax_4^k}{x_1^k} \cdot u^k + C_2 \cdot (u^k)^2) \\ x_2^k + T_s(-x_1^k x_4^k + C_3 \frac{x_2^k - bx_4^k}{x_1^k} + C_2 \frac{x_2^k + ax_4^k}{x_1^k}) & -T_s C_2 \cdot u^k \\ x_3^k + T_s x_4^k & + 0 \\ x_4^k + T_s(C_4 \frac{x_2^k + ax_4^k}{x_1^k} - C_5 \frac{x_2^k - bx_4^k}{x_1^k}) & -T_s C_4 \cdot u^k \\ x_5^k + T_s(x_1^k \cos(x_3^k) - x_2^k \sin(x_3^k)) & + 0 \\ x_6^k + T_s(x_1^k \sin(x_3^k) + x_2^k \cos(x_3^k)) & + 0 \end{bmatrix}.$$

This representation can be written more generally as

$$x^{k+1} = g(x^k, u^k, T_s) \quad \forall k \in \{0, \dots, T-1\}. \quad (5.1)$$

Treating T_s as a constant, the above expression can then be reformulated as Equation (3.1).

The cost function is defined as

$$J_L(\mathbf{x}, \mathbf{u}) = \frac{1}{2}w_1 \sum_{k=0}^T \left[\left\| P_x^k - P_{x,\text{ref}}^k \right\|_2^2 + \left\| P_y^k - P_{y,\text{ref}}^k \right\|_2^2 \right] + \frac{1}{2}w_2 \sum_{k=0}^T (u^k)^2,$$

where $w_1 \in \mathbb{R}^{1 \times 2}$ and $w_2 \in \mathbb{R}$ are weight factors that together form the weight vector $w \in \mathbb{R}^3$. The first term drives the vehicle along a reference trajectory and the second term limits control effort. Since this objective is quadratic, it can be rewritten in the form shown in Equation (3.2).

Observe that the trajectory optimization problem with a vehicle as mobile agent has a quadratic objective and nonlinear equality constraints. Therefore, this problem fits well with the problem formulation in (3.3), and can be solved effectively using the proposed method. Note that the infinity norm is used to construct the trust region constraints, because it has a linear implementation.

5.1.2 Initialization

Given no prior information on what inputs will solve the problem, some reasonable input (e.g., zero input) and trajectory pair $(\mathbf{x}_0, \mathbf{u}_0)$ is selected to cold start the SQP iterations. By numerically integrating the system to obtain the state trajectory, we can ensure the trajectory satisfies the equality constraints.

5.1.3 Extension to Collision Avoidance

With an extension to also consider obstacle avoidance, additional inequality constraints are needed. Therefore, we redefine the new vector of inequality constraints by h . Recall that the initial input and trajectory pair $(\mathbf{x}_0, \mathbf{u}_0)$ is selected to cold start the SQP iterations, but it is possible that $(\mathbf{x}_0, \mathbf{u}_0)$ violates the inequality constraints. In that case, the trust region condition will lead to infeasibility, as Δ may not be large enough to allow \mathbf{u} to be modified so that \mathbf{h} becomes nonpositive. Similar constraint violations could occur at any iteration.

To handle this potential infeasibility, we propose to relax the inequality constraints as follows.

$$\mathbf{h}_s + \left. \frac{dH}{d\mathbf{u}} \right|_s (d\mathbf{u})_s \leq \mathbf{r}_s, \quad (5.2)$$

where $\mathbf{r}_s \in \mathbb{R}^p$ is the constraint violation vector in the s -th iteration, whose entries $(r_i)_s \in \mathbb{R}$ are variables that measure the amount of constraint violation. For inequality constraints that cannot be relaxed (i.e. the original input and state bounds), the corresponding r_i 's are set to zero.

Meanwhile, the constraint violation must be reduced to zero before the algorithm converges. To achieve this, \mathbf{r}_s is introduced into the following objective function:

$$J_r(\mathbf{r}_s) = \rho \mathbf{r}_s^\top \mathbf{r}_s,$$

where $J_r : \mathbb{R}^p \rightarrow \mathbb{R}$ evaluates the cost of constraint violation and $\rho \in \mathbb{R}_{\geq 0}$ is the weight coefficient. This cost metric is quadratic and thus satisfies the assumption that the objective is quadratic. The weight ρ is increased steadily as the number of SQP iterations increases, forcing inequality constraints to be prioritized in optimization and constraint violation to be eventually eliminated.

Finally, to implement collision avoidance constraints, we consider the case of static obstacles with known position and dimension. By modeling these obstacles as ellipses, the following constraints are constructed to prevent collision:

$$\frac{(P_x^k - C_x)^2}{A^2} + \frac{(P_y^k - C_y)^2}{B^2} \geq 1,$$

where C_x and C_y are coordinates of the center of the ellipse that models an obstacle, A and B are the major and minor axes of the ellipse [45]. Such constraints are nonlinear and nonconvex, and will be reformulated into Equation (5.2). Here we choose ellipses as an example, while other less regular shapes with collision avoidance constraints that are differentiable with respect to the states are also acceptable.

5.2 Motion Planning Problem: Simulation

To demonstrate the performance of SQP-TS, we compare our method with the traditional trust region SQP with first-order constraint linearization, which we denote by LSQP. We make this comparison to capture the impact of the trajectory sensitivities and the benefits of using SQP-TS. In addition, we also show how trajectory sensitivities lead to improved robustness in very nonlinear, challenging scenarios.

In all subsequent test cases, unless otherwise defined, it is assumed that the initial vehicle position (P_x^0, P_y^0) is the same as the initial position of reference, the initial vehicle

heading θ^0 is selected to be 0 rad , and the initial trajectory is chosen to be the zero input trajectory evaluated using (5.1). The vehicle parameters are fixed for all test cases and are set to

$$\begin{aligned} C_{\alpha f} = C_{\alpha r} &= -50000 \text{ N/rad}, C_{x f} = C_{x r} = 5000 \text{ N}, \\ s_f = s_r &= 0.1, a = b = 1.2 \text{ m}, \\ m &= 1200 \text{ kg}, I_z = 2000 \text{ kg} \cdot \text{m}^2. \end{aligned} \tag{5.3}$$

5.2.1 Convergence behaviour and Trust Region Radius

Fix start and end points, Dubins paths of variable turning radius are generated using the Python *Dubins* package. We select $(P_x^0, P_y^0, \theta^0) = (100, 100, 0.5)$ and $(P_x^T, P_y^T, \theta^T) = (105, 105, 2.5)$ where the unit is meters for coordinates and radians for angle. The minimum turning radius of the path is randomly sampled from the range of $(1, 6)$ radians with a step size of 0.3. Using these Dubins paths as references, both **SQP-TS** and LSQP are applied to solve the vehicle trajectory optimization problem. The relevant parameters used in solving this test case are

$$\begin{aligned} T &= 80 \text{ s}, T_s = 0.01 \text{ s}, \tau = 1.0, \\ w_1 &= 100, w_2 = 20, w_3 = 0, \end{aligned}$$

where τ is the tolerance for the change in vehicle paths of two subsequent iterations, and the algorithm terminates when τ is not exceeded. Another important parameter is the radius of the trust region, Δ , whose magnitude will be actively adjusted based on the performance of both algorithms.

A total of 60 Dubins paths are generated as reference paths. As **SQP-TS** and LSQP are used to perform trajectory optimization over these references, it is observed that the maximum allowable Δ without divergence is 0.5 for **SQP-TS**, and 0.3 for LSQP. Therefore, we first set $\Delta = 0.3$ to run both algorithms, and then set $\Delta = 0.5$ to run **SQP-TS**. To make a comparison, we record the total iterations to converge and collect them in the box plot in Figure 5.1. The numbers in the x labels represent the magnitude of Δ .

When both LSQP and **SQP-TS** are executed at $\Delta = 0.3$, **SQP-TS** takes significantly less number of iterations to converge than LSQP. When **SQP-TS** is run at $\Delta = 0.5$, its convergence iterations are slightly reduced, compared to the same algorithm run at $\Delta = 0.3$. The fact that **SQP-TS** converges in less iterations given the same trust region radius and is allowed a larger trust region radius without divergence is contributed by the trajectory-sensitivities-based linearization of constraints of problem (3.3) and its improvement in accuracy of approximation. With increased trust region radius, \mathbf{u} is able to converge in

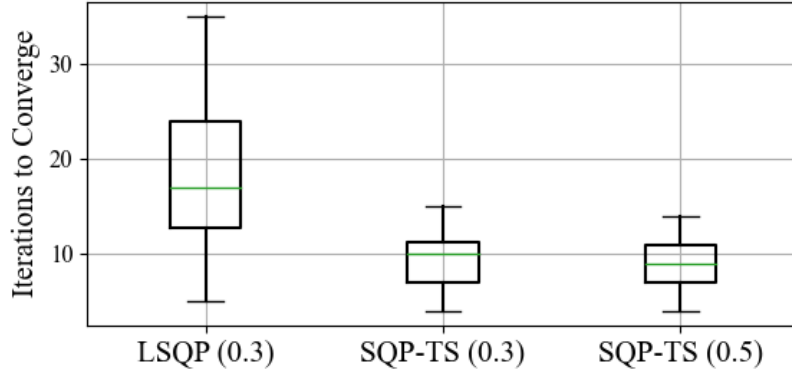


Figure 5.1: Total iterations to converge. LSQP is shown for a trust region of 0.3, while SQP-TS is shown for a trust region of both 0.3 and 0.5.

less number of iterations. The computation time per iteration is discussed in the next section.

5.2.2 Convergence behaviour and Initial Inputs

Using 20 of the Dubins paths generated in the previous test case, we want to explore how initial inputs affect the performance of both methods. The initial trajectories are calculated by numerical integration of Equation (5.1) with the following choice of constant inputs.

$$u^k = 0, 0.1, 0.2, 0.3 \text{ rad/s}^2, \quad \forall k \in \{0, \dots, T-1\}.$$

Setting $\Delta = 0.3$, $\tau = 1$, and executing both SQP-TS and LSQP, the iterations to converge and total solver time to converge are recorded. The solver time is the time used to solve each optimization subproblem reported by Gurobi and does not account for the time spent building the problem model. Note that both SQP-TS and LSQP spend 0.03 s per iteration to perform constraint linearization, while SQP-TS computes trajectory sensitivities and LSQP computes gradients. Therefore, both methods spend a similar amount of time constructing the subproblem. The results are shown in Figure 5.2.

As the initial input changes, the iterations to convergence and the total solver time of SQP-TS are relatively constant, while those of the LSQP algorithm change drastically. For the extreme case of $u^k = 0.3$ for each k , LSQP takes about 11 times of iterations to converge than SQP-TS, and about 6 times of solver time. Therefore, SQP-TS is not as sensitive to initial inputs as LSQP is and has consistent performance. Moreover, the

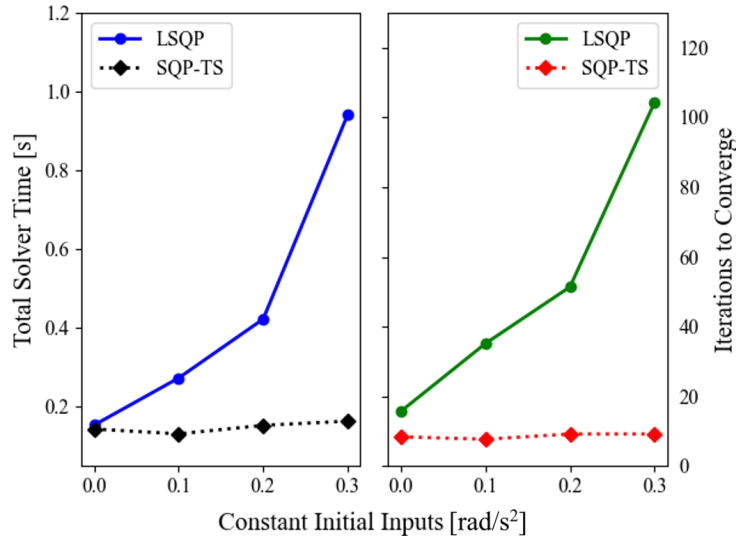


Figure 5.2: Total solver time (blue and black lines) and iterations to converge (green and red lines) for different constant initial inputs using LSQP (solid lines) and SQP-TS (dotted lines) methods.

fact that **SQP-TS** works with zero input initialization demonstrates its good convergence behaviour. To further improve convergence speed, the algorithm can be warm started with trajectories computed using (5.1) with \mathbf{u} generated by other controllers, such as a linear model predictive controller (LMPC) [10].

5.2.3 Example with Collision Avoidance

Now we extend the trajectory optimization problem to include static obstacles. First, a simple scenario is considered in which the reference path is a straight line passing through an obstacle. This test scenario is adapted from the example in [45]. The following parameters are used.

$$\begin{aligned} \Delta &= 0.3 \text{ rad/s}^2, \quad \tau = 0.3, \quad \rho = (1000)(s), \\ T &= 80 \text{ s}, \quad T_s = 0.02 \text{ s}, \quad w_1 = 1000, \quad w_2 = 1, \quad w_3 = 0, \end{aligned}$$

where the weight coefficient of the collision cost ρ is a variable proportional to the number of past algorithm iterations s . For this example, we set the initial vehicle heading to align with the reference path. Using LSQP and **SQP-TS** to solve this problem, the final vehicle paths are plotted in Figure 5.3.

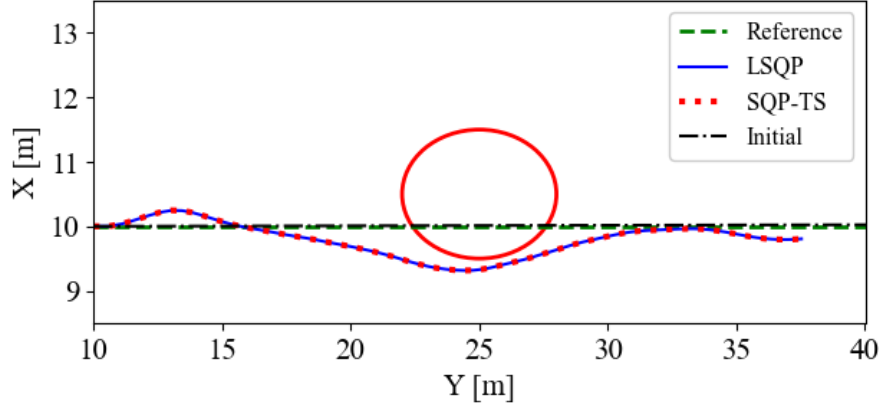


Figure 5.3: Straight reference path with an obstacle in which both the proposed method and LSQP converge.

A second reference path is selected to be a Dubins curve generated using the same approach as in the previous section, which intersects a static obstacle. The same set of parameters are used except for $T_s = 0.05$ s, and LSQP and SQP-TS are applied to solve this problem. The final vehicle paths are plotted in Figure 5.4, where we see that both algorithms converge to the same final paths. For the straight reference path, LSQP takes 80 iterations to converge, while SQP-TS takes 12 iterations and converges faster. For the Dubins reference, both methods take 8 iterations to converge.

Finally, consider a complex, nonsmooth reference path composed of straight lines, curves, sharp turns, and which passes through obstacles. The zero input initialization is an infeasible path that crosses an obstacle(s). The same set of parameters as the straight-line test scenario is used, with the following changes.

$$\Delta = 0.02 \text{ rad}^2/\text{s}, \tau = 4, T = 200 \text{ s}, T_s = 0.1 \text{ s}.$$

A larger τ is used because the length of the path is longer than the straight line or the Dubins path. A smaller trust region radius is selected because the reference path is more complex and more accurate approximation is needed. For this test scenario, LSQP fails due to numerical error at 100 iterations. SQP-TS converges in 14 iterations, and the final vehicle path is shown in Figure 5.5.

Closer investigation reveals that the numerical error is likely caused by approximation errors in the states that impact the collision avoidance constraints. In LSQP, this eventually leads to constraint violation and causes numerical trouble. Multiple different obstacle

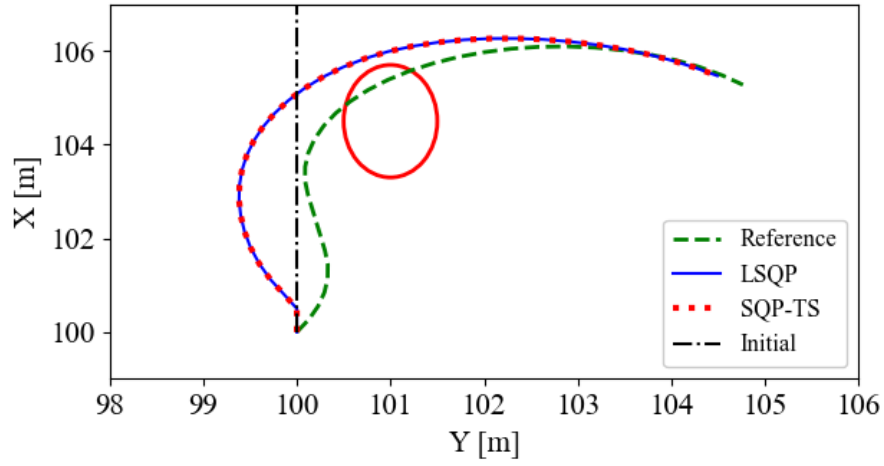


Figure 5.4: Dubins reference path with an obstacle in which both the proposed method and LSQP converge.

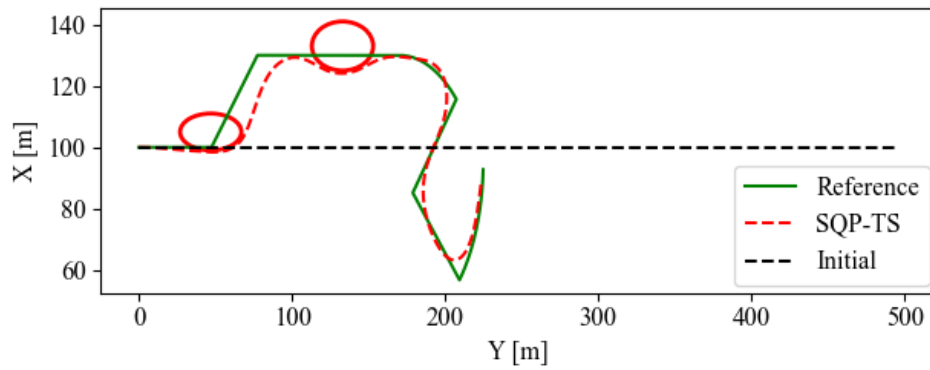


Figure 5.5: Complex path with multiple obstacles in which SQP-TS converges but LSQP fails.

arrangements have been tested, but LSQP always fails due to the same issue. On the other hand, [SQP-TS](#) solves the optimization problem with a more accurate approximation, and no such failure has been observed. Therefore, [SQP-TS](#) is better at handling collision avoidance constraints with a complex reference path than the LSQP method, and is not as prone to convergence failure.

5.3 Driving Style Imitating Problem: Setup

Now that numeric examples of how the [LL](#) motion planning problem is solved and available, we then demonstrate how to approach the driving style imitating problem using numeric examples. By solving this [UL](#) of the [BP](#) problem, the personalized [AV](#) motion planning task is effectively achieved. The proposed projected subgradient update algorithm is implemented in Python.

5.3.1 Dynamic bicycle Model

In [Chapter 5.1](#), a single-track bicycle model of the vehicle is used. This model only takes steering angle as input, thus highly limits the types of control we have over the vehicle. To enable better control of an [AV](#), we consider a dynamic bicycle model of vehicles, which takes the following set of state variables and system inputs:

$$x = [v_x \ v_y \ \theta \ \dot{\theta} \ P_x \ P_y]^\top, \quad \bar{u} = [\delta \ a_x],$$

where the state vector is the same as that of a single-track bicycle model, $\bar{u} \in \mathbb{R}^2$ is the input vector, δ is the steering angle, and a_x is the longitudinal acceleration. Realistically, δ is controlled by the steering wheel and a_x is controlled by the combined effect of the accelerator and brake pedals.

The [CT](#) dynamic bicycle model is described in [ODE](#) form as below [\[29\]](#).

$$\dot{x} = \begin{bmatrix} v_y \dot{\theta} + a_x \\ -v_x \dot{\theta} - \frac{C_{\alpha_f}}{m} \frac{v_y + a\dot{\theta}}{v_x} - \frac{C_{\alpha_r}}{m} \frac{v_y - b\dot{\theta}}{v_x} + \frac{C_{\alpha_f}}{m} \delta \\ \dot{\theta} \\ \frac{1}{I_z} \left(-aC_{\alpha_f} \frac{v_y + a\dot{\theta}}{v_x} + bC_{\alpha_r} \frac{v_y - b\dot{\theta}}{v_x} + aC_{\alpha_f} \delta \right) \\ v_x \sin \theta + v_y \cos \theta \\ v_x \cos \theta - v_y \sin \theta \end{bmatrix}.$$

For simplicity, we define some constant terms.

$$C_1 = \frac{C_{\alpha_f}}{m}, \quad C_2 = \frac{C_{\alpha_r}}{m}, \quad C_3 = \frac{aC_{\alpha_f}}{I_z}, \quad C_4 = \frac{bC_{\alpha_r}}{I_z}, \quad (5.4)$$

then the **CT** dynamic bicycle model can be discretized using the sampling time T_s to be

$$x^{k+1} = \begin{bmatrix} x_1^k + T_s(x_2^k x_4^k) & + T_s u_2^k \\ x_2^k + T_s(-x_1^k x_4^k - C_1 \frac{x_2^k + ax_4^k}{x_1^k} - C_2 \frac{x_2^k - bx_4^k}{x_1^k}) & + T_s C_1 u_1^k \\ x_3^k + T_s x_4^k & + 0 \\ x_4^k + T_s(-C_3 \frac{x_2^k + ax_4^k}{x_1^k} + C_4 \frac{x_2^k - bx_4^k}{x_1^k}) & + T_s C_3 u_1^k \\ x_5^k + T_s(x_1^k \cos(x_3^k) - x_2^k \sin(x_3^k)) & + 0 \\ x_6^k + T_s(x_1^k \sin(x_3^k) + x_2^k \cos(x_3^k)) & + 0 \end{bmatrix}. \quad (5.5)$$

Treating T_s as a constant, the **DT** system can be reformulated as Equation (3.1).

5.3.2 Objective and Subgradient

We recall that the objective function for the driving style imitating problem is defined as

$$J_U(\mathbf{y}_d, \mathbf{u}_*(w)) = \frac{1}{2} \sum_{d=1}^{n_d} \|\mathbf{y}_d - \mathbf{y}(\mathbf{u}_*(w))\|^2,$$

which suggests that the x and y positions of a vehicle are equally important when we attempt to learn a demonstration. To execute Algorithm 2, we need to compute the subgradients of the **UL** objective with respect to w (i.e. the **UL** subgradients). The calculation procedure is introduced in Chapter 4.2 and the following expression is derived.

$$\frac{dJ_U}{dw} = \frac{\partial J_U}{\partial \mathbf{y}} \bar{C} \frac{\partial \mathbf{x}}{\partial \mathbf{u}_*} \frac{dF}{dw}.$$

Given J_U , the first term is obvious.

$$\frac{\partial J_U}{\partial \mathbf{y}} = [1 \quad 1 \quad \dots \quad 1] \in \mathbb{R}^{2T}.$$

The second term is first-order **TS** and can be computed following Chapter 3.1.2; the third term is obtained from a sensitivity analysis of the **LL** motion planning problem discussed in Chapter 4.1.

The weight factors defined for the **LL** problem include w_1 for the path tracking objective and w_2 for the minimum control objective. We assume that for each demonstration the test driver maintains the same driving style. In other words, w_1 and w_2 are fixed for $k \in \{0, \dots, T - 1\}$, hence their dimensionalities are given by

$$w_1 \in \mathbb{R}^{1 \times 2}, \quad w_2 \in \mathbb{R}^{1 \times 2}.$$

Note that the dimension of w_2 is bigger than before as we are now working with a two-input system. Therefore, the input weight vector for the **LL** problem is

$$w = [w_1 \quad w_2] \in \mathbb{R}^{1 \times 4}.$$

5.3.3 Initialization and Parameter Selection

Given that there is no prior information on what weights will solve the problem, some reasonable weight vector w_0 is selected to cold start the projected subgradient update iterations. Examples of such weight vectors include

$$(w)_0 = [1 \quad 1 \quad 1 \quad 1 \quad 1], \quad (w)_0 = [1 \quad 1 \quad 10 \quad 10 \quad 10], \quad (w)_0 = [10 \quad 10 \quad 1 \quad 1 \quad 1],$$

where the bracket followed by a subscript i indicates the weight in i -th subgradient update iteration.

When solving the **LL** motion planning problem in Chapter 5.2, the **SQP-TS** algorithm is cold-started with zero initialization given no prior information of a suitable input trajectory. However, when solving the **UL** imitating problem, which requires the **LL** problem to be solved first to compute the subgradients, the **SQP-TS** algorithm can be warm started. This is because Algorithm 2 is iterative and the **LL** problem is solved at each iteration to produce an optimal input trajectory u_* given some input weight. When the weight vector is updated for the next iteration, the optimal vehicle path changes, but this input trajectory still produces a vehicle path that is relatively close to the new optimal path. Therefore, we can warm-start the **SQP-TS** algorithm using the optimal input trajectory u_* calculated in the previous projected subgradient update iteration.

The projected subgradient update algorithm has many parameters and tuning them affects the performance of the algorithm. The relevant parameters are given in Table 5.1.

Table 5.1: Parameters of projected subgradient update algorithm

α_0	Initial step length	Δu_1	LL trust region radius for u_1
β	Decrement factor of step length	Δu_2	LL trust region radius for u_2
γ	Tolerance of constant step length	ξ_1	LL tolerance of convergence for u_1
ϵ	UL tolerance of convergence	ξ_2	LL tolerance of convergence for u_2

We highlight the two parameters β and γ . As shown in Algorithm 2, the subgradient is updated by some fixed step length α_0 , before the absolute value of the change in the UL objective falls below the tolerance γ . Then, the step length is decreased by a decrement factor of β . Mathematically, this procedure is equivalent to:

$$\alpha_s = \begin{cases} \alpha_{s-1} & \text{if } \Delta J_U \geq \gamma \\ \beta \alpha_{s-1} & \text{if } \Delta J_U < \gamma. \end{cases} \quad (5.6)$$

It is also important to note that with a fixed step length, the change in UL cost is not guaranteed to decrease when approaching a local minimum. In some cases, this cost may oscillate around some value with considerable difference, in which case decrement in step length is not invoked and the algorithm never converges. To also take this situation into account, we introduce the average cost defined as follows.

$$J_{\text{avg}} = \frac{1}{\tilde{s}} \sum_{i=1}^{\tilde{s}} (J_H)_{s-i},$$

where \tilde{s} is the number of costs to take the average of and s is the index of the current projected subgradient update iteration. For our simulation, we select $\tilde{s} = 10$. When the algorithm fails to converge using the vanishing step length scheme in (5.6), we opt for the following scheme:

$$\alpha_s = \begin{cases} \alpha_{s-1} & \text{if } \min\{\Delta J_U, J_{\text{avg}}\} \geq \gamma \\ \beta \alpha_{s-1} & \text{if } \min\{\Delta J_U, J_{\text{avg}}\} < \gamma. \end{cases} \quad (5.7)$$

Suitable values of the parameters in Table 5.1 are determined by some tests. Generally, for complex reference paths, smaller values of α_0 , Δu_1 and Δu_2 result in better estimations of UL and LL subgradients and improved performance.

5.4 Driving Style Imitating Problem: Simulation

In this section, we present the simulation results of driving style imitation using the projected subgradient update algorithm. We investigate the performance of this algorithm under different testing conditions. The reference paths used include a smooth, curved path and a nonsmooth, complex path as shown in Figure 5.5. We consider both the case where all demonstrations follow the same weight vector and where similar but different weight vectors exist. We also explore the obstacle avoidance behaviour when reference paths include obstacles at various locations. The objective of these simulations is to verify the ability of the proposed algorithm to accurately imitate a driver’s driving style under a variety of scenarios.

We simulate drivers’ driving data to serve as demonstrations. To do so, we consider the same vehicle dynamics as shown in (5.5) with vehicle parameters defined in (5.3) and (5.4). Given a reference trajectory and obstacle information and choosing a weight vector w that imitates a driver’s driving style, we solve the LL vehicle motion planning problem using SQP-TS to obtain some vehicle state trajectory that balances the LL cost metrics of reference tracking, control effort minimization, and jerk reduction. We then extract positional information of the vehicle from the state trajectory, which can be used as a demonstration \mathbf{y}_d .

When choosing the weight vector w , we make some reasonable assumptions to simplify the problem. First, we assume that a driver’s driving style is consistent within the observation period. This means that for $k \in \{0, 1, \dots, T\}$, w is a constant vector. We also assume that a driver maintains similar driving styles when passing through the same path in similar environments. This suggests that the weight vector w of the same driver does not vary too much from one demonstration to another, given similar reference paths and obstacle arrangements. To simulate multiple demonstrations for the same target driver, we change the reference path and obstacle positions by an acceptable amount, and resolve the LL problem. This can be done by sampling from a subset of possible reference paths and obstacle positions. To take into consideration small variations in the driver’s data, we also sample w from a neighborhood of the nominal weight.

For the LL vehicle motion planning problem, unless otherwise specified, we assume that the initial vehicle position (P_x^0, P_y^0) is equal to the initial position of the reference path, the initial vehicle heading θ^0 is 0 rad , and the initial vehicle trajectory is chosen to be the zero input trajectory evaluated using (5.5).

5.4.1 Reference Tracking

Consider the scenario of a driver driving on the road based on a planned reference path that could be produced by a navigation app. Despite the reference path being the same, each driver produces different driving data based on their own driving style. In order to imitate the driving style of a target driver, we apply the projected subgradient update algorithm to find a suitable w . In this section, we consider two types of reference path: a smooth, curved path and a nonsmooth, complex path.

The curved reference path is generated using the dynamic bicycle model (5.5) with the following parameters:

$$x^0 = [16.67 \ 0 \ 0 \ 0 \ 100 \ 100]^\top, \quad \mathbf{u} = \begin{bmatrix} u_\delta \\ 3 \end{bmatrix}, \quad T = 80, \quad T_s = 0.01 \text{ s},$$

where $u_\delta \in \mathbb{R}$ is a series of steering inputs whose magnitude varies from demonstration to demonstration. We first consider the simple case of a single demonstration. Choose $u_\delta = 0.3$ and let the driving habit of the simulated driver be $w_{\text{ref}} = [1 \ 1 \ 10 \ 1]$, a sample demonstration is created. To run the projected subgradient update algorithm on this demonstration, we choose the initial weight $(w)_0 = [1 \ 1 \ 50 \ 50]$, which produces a vehicle trajectory that is very different from the demonstration. We also choose the following parameters:

$$\begin{aligned} \alpha_0 &= 0.1, \quad \beta = 0.01, \quad \gamma = 0.8, \quad \epsilon = 0.00001, \\ \Delta u_1 &= 0.05, \quad \Delta u_2 = 0.05, \quad \xi_1 = 0.01, \quad \xi_2 = 0.001. \end{aligned}$$

The algorithm converges in 48 iterations. The simulation results are shown in Figure 5.6. In this figure, we see that the output of the LL vehicle controller is initially away from the demonstration. After solving the BP to find w , the new controller output matches the demonstration very closely.

Now we move on to a more complex case of multiple demonstrations. We vary the reference paths by choosing

$$u_\delta = \{0.1, 0.15, 0.2, 0.25, 0.3, 0.35\},$$

where each input value gets used twice except for the first and final values, and vary the weight vector by sampling from the following ranges:

$$w_{\text{ref}} = [w_{r0} \ w_{r1} \ w_{r2} \ w_{r3}], \quad w_{r0} = w_{r1}, \quad w_{r0}, w_{r3} \in [0.95, 1.05], \quad w_{r2} \in [9.5, 10.5].$$

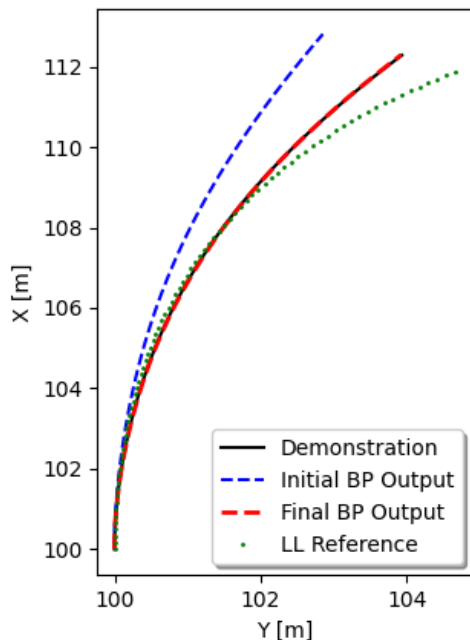


Figure 5.6: Personalized vehicle motion planning following a curved reference path using one demonstration.

The exact values of the weight vectors can be found in Appendix B.2. Combining the two, we get a total of 10 demonstrations. The choice of inconsistent weights closely resembles real-life situations where a driver exhibits similar but not exactly the same driving style. We use the same initialization of weights and algorithm parameters as in the single demonstration case and run projected subgradient update. In addition, we include J_{avg} to assess whether the step length should be decremented. The algorithm converges in 78 iterations, and the simulation results are plotted in Figure 5.7.

Observations similar to those in Figure 5.6 can be made by looking at Figure 5.7. We see that the weight found using the BP program produces vehicle motion planning results that closely match the demonstration. In this figure, the reference paths used in the LL vehicle motion planning problem are omitted for clarity. A comparison of the demonstrations and the reference paths provided can be found in Appendix B.3.

The complex reference path is the path used in Chapter 5.2.3, which is a combination of straight lines, smooth turns, and sharp turns. Similar to the previous example of a

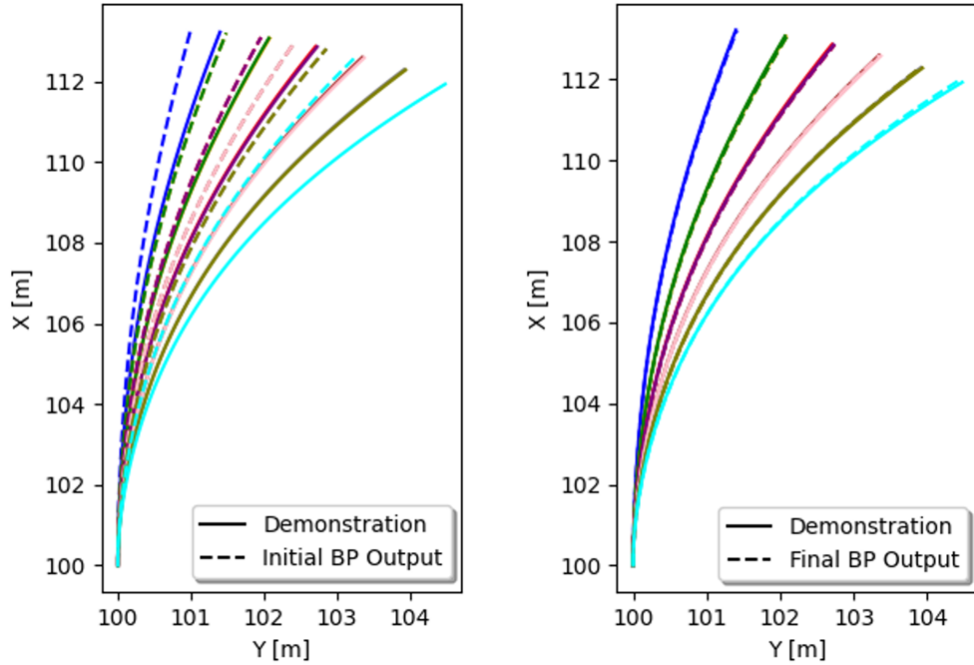


Figure 5.7: Personalized vehicle motion planning following a curved reference path using 10 demonstrations.

curved path, we first consider the simple case of a single demonstration generated using $w_{\text{ref}} = [1 \ 1 \ 10 \ 1]$. With an initial weight of $(w)_0 = [1 \ 1 \ 200 \ 200]$ and the set of parameters

$$\alpha_0 = 2, \quad \beta = 0.5, \quad \gamma = 0.2, \quad \epsilon = 0.00001, \\ \Delta u_1 = 0.05, \quad \Delta u_2 = 0.05, \quad \xi_1 = 0.5, \quad \xi_2 = 0.2,$$

the projected subgradient update algorithm converges in 48 iterations. The simulation results are shown in Figure 5.8. The figure on the left of Figure 5.8 shows how the projected subgradient algorithm generates w whose resultant vehicle path matches the simulated driver’s data well. The reference path is separately plotted on the right for readability.

Next, we consider the more complex case of multiple demonstrations. We vary the complex reference path by modifying the input trajectory to the vehicle model, and also vary the weight vector by sampling as described earlier. The sampled weights are recorded in Appendix B.2. Again, 10 demonstrations are generated. Let the initial weight and algorithm parameters be the same as used in the single demonstration case, convergence is observed in 73 iterations, and the simulation results are plotted in Figure 5.9 and Figure 5.10, where the 10 demonstrations are plotted separately.

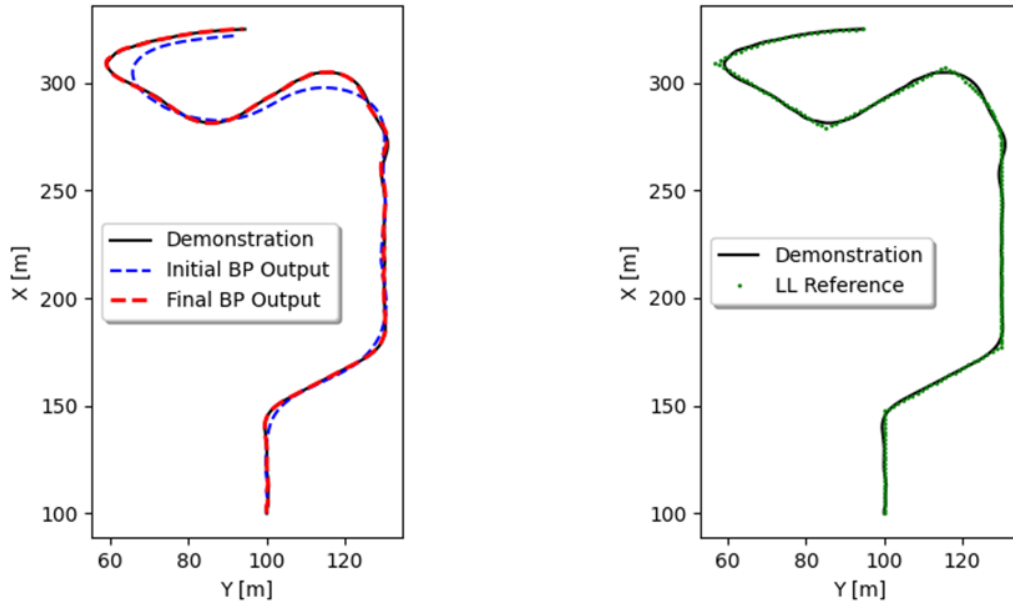


Figure 5.8: Personalized vehicle motion planning following a complex reference path using one demonstration.

In Figure 5.9 and Figure 5.10, the plot on the left shows the path of the vehicle generated using $(w)_0$, and the plot on the right shows the path of the vehicle generated using w (i.e. the result of solving the BP problem). Again, we see that the demonstrations are accurately followed by running our proposed controller with small deviations. The reference path has been omitted for readability; see Appendix B.3 for its plots.

5.4.2 Obstacle Avoidance

Consider the scenario of a driving driving past a roundabout, but an obstacle (e.g., some trash, a puddle, a vehicle parked due to malfunction) is on the path and the driver avoids it when driving. The reference path in this case is simply the center line of the roundabout. To track this reference path while avoiding the obstacle that passes through it, different drivers take different paths. In this section, we consider the driving-style imitating problem of this obstacle avoidance task on a roundabout.

To generate a reference path that resembles the centerline of a roundabout, we use the

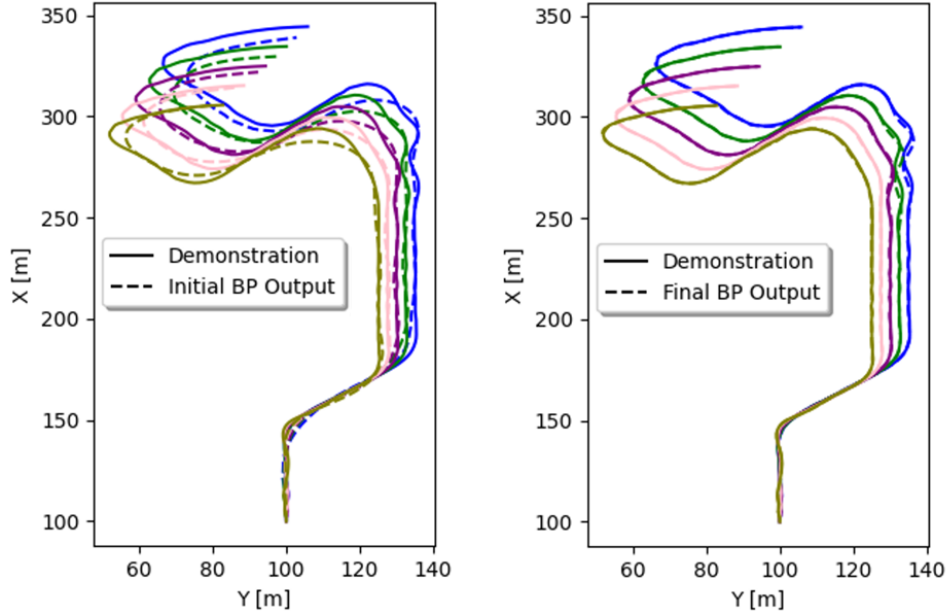


Figure 5.9: Personalized vehicle motion planning following a complex reference path using the first 5 of 10 demonstrations.

dynamic bicycle model (5.5) with the following parameters:

$$x^0 = [16.67 \ 0 \ 0 \ 0.8 \ 100 \ 100]^\top, \quad \mathbf{u} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad T = 80, \quad T_s = 0.05 \text{ s}.$$

In addition, we consider two obstacles modeled as ellipses with

$$\text{obs}_1 = [3 \ 3 \ 119.9 \ 130.1], \quad \text{obs}_2 = [3 \ 3 \ 117.9 \ 130.1],$$

where the first and second entries of the vector are the radii of the ellipse along the x and y directions, and the third and fourth entries are the x and y positions of the center of the ellipse. Both obstacles intersect the reference path, while the majority of obs_1 is located in the positive x direction of the reference, and the majority of obs_2 lies in the negative x direction of the reference. In other words, to avoid obs_1 or obs_2 while following the reference path, one would take a detour from the opposite sides of the obstacle.

Trajectories that avoid an obstacle from opposite sides fall under different topological groups, and it is difficult to change the groups of a vehicle path using trajectory optimization. To address this, we initialize the LL motion planning problem with an input

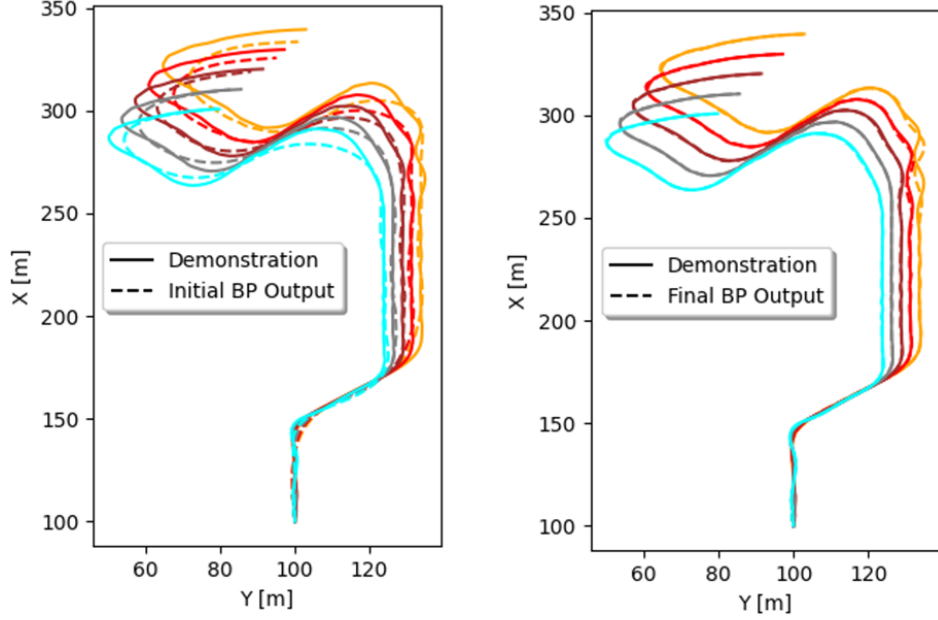


Figure 5.10: Personalized vehicle motion planning following a complex reference path using the last 5 of 10 demonstrations.

trajectory that results in a state trajectory that is in the same topological group as the demonstration. Specifically, for obs_1, we use the zero input trajectory; for obs_2, we set:

$$\delta[0 : 10] = 0.49, \delta[10 : 50] = 0, \delta[50 : 60] = 0.4, \delta[60 : 80] = 0, a_x = 0.$$

. Similarly to the previous example, we first consider the simple case of a single demonstration. For obs_1, we consider $w_{\text{ref}} = [1 \ 1 \ 1 \ 1]$ to generate the demonstration. Then, we choose the following parameters

$$\begin{aligned} \alpha_0 &= 0.1, \beta = 0.8, \gamma = 0.01, \epsilon = 0.0001, \\ \Delta u_1 &= 0.05, \Delta u_2 = 0.05, \xi_1 = 0.32, \xi_2 = 0.02. \\ (w)_0 &= [1 \ 1 \ 50 \ 50], \end{aligned}$$

and apply the projected subgradient update algorithm to solve for w . The algorithm converges in 318 iterations. Likewise, for obs_2, we use $w_{\text{ref}} = [10 \ 10 \ 1 \ 1]$ to generate the demonstration, and use the same initial weight and parameters to solve the problem. The algorithm converges in 249 iterations. Visual representations of results with both obstacle positions can be found in Figure 5.11 and Figure 5.12.

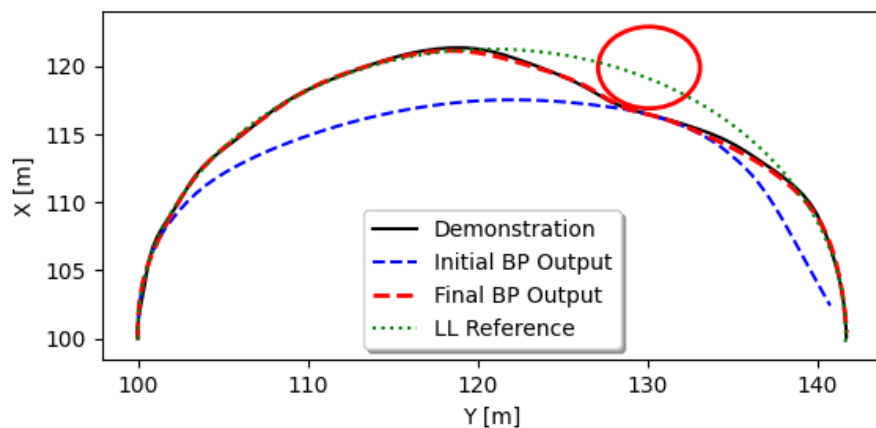


Figure 5.11: Personalized vehicle motion planning following a curved reference path using one demonstration.

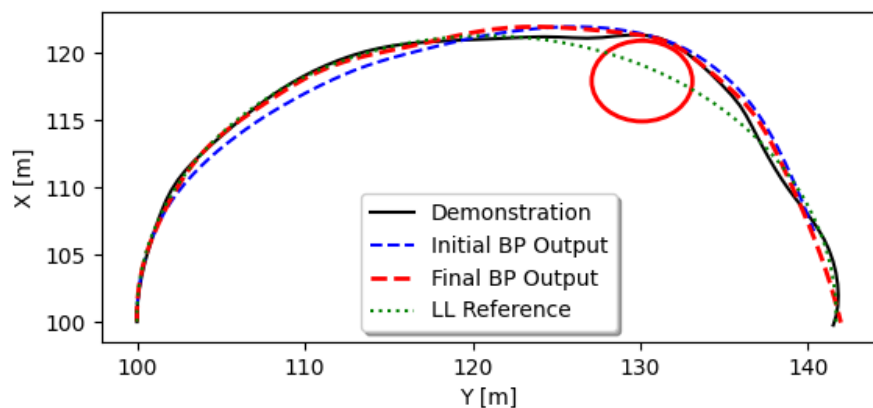


Figure 5.12: Personalized vehicle motion planning following a curved reference path using one demonstration.

Both figures demonstrate how the solution of the BP problem effectively imitates the driver’s driving style and generates vehicle paths that match the demonstrations. In addition, we note that the vehicle path with $(w)_0$ differs greatly from the vehicle path with w for obs_1, and the improvement is more significant. In contrast, the two vehicle paths for obs_2 are similar at initialization, so the improvement is not as obvious. Nevertheless, the UL costs J_H indeed decrease.

We then consider the more complex case of multiple demonstrations. To do so, we shift

the obstacle positions by randomly sampling the following set:

$$x_{\text{obs}} \in [\tilde{x} - 0.5, x - 2.5] \cup [\tilde{x} + 0.5, x + 2.5], \quad y_{\text{obs}} \in [110, 130]$$

where \tilde{x} is the x position of the reference path given the sampled y_{obs} . This ensures that the reference path always passes through the obstacle while the center of the ellipse never lies on the reference path, which may result in singularities. We also randomly sample the driver’s driving style vector as follows:

$$w_{\text{ref}} = [w_{r0} \quad w_{r1} \quad w_{r2} \quad w_{r3}], \quad w_{r0} = w_{r1}, \quad w_{r2} = w_{r3}, \quad w_{r0}, w_{r3} \in [0.95, 1.05].$$

A total of 10 demonstrations are generated. The exact locations of obstacles and weight vectors can be found in Appendix B.2. We then initialize with $(w)_0 = [1 \quad 1 \quad 50 \quad 50]$ and solve the BP problem using the same parameters as in the single demonstration case, except for using $\alpha_0 = 0.5$ and $\gamma = 0.2$. The algorithm converges in 205 iterations. The simulation results are plotted in Figure 5.13, Figure 5.14, and Figure 5.15.

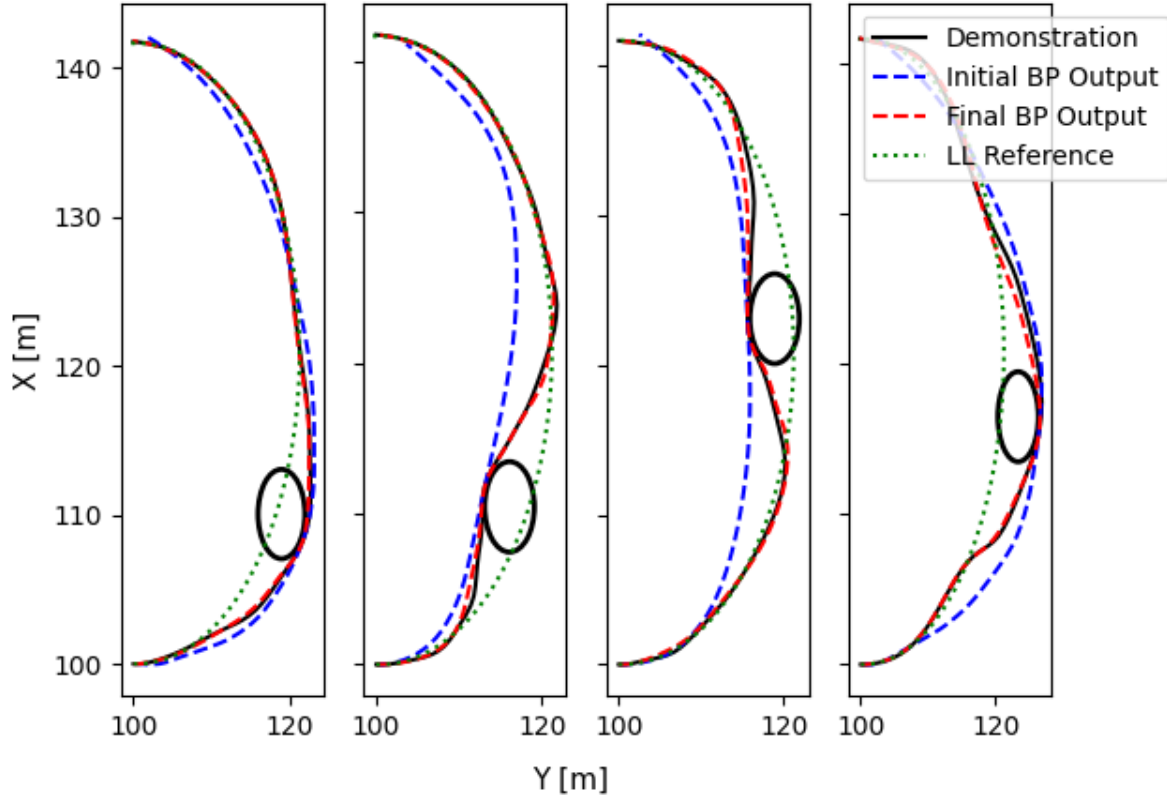


Figure 5.13: Personalized vehicle motion planning at a roundabout with obstacle using Demonstrations 1-4.

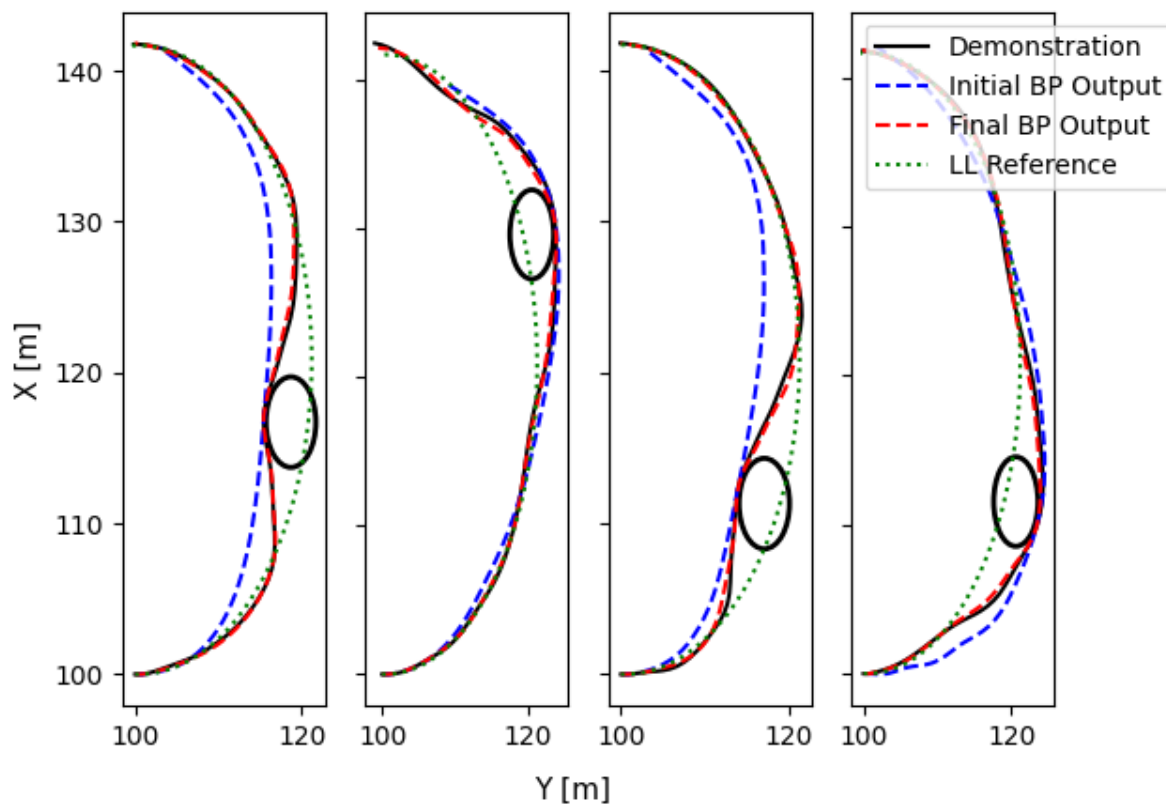


Figure 5.14: Personalized vehicle motion planning at a roundabout with obstacle using the first Demonstrations 5-8.

Plots of 10 demonstrations have been separated into 3 figures to increase figure clarity. In each figure, there are 2 or 4 subplots; in each subplot, the demonstration, the vehicle path generated using $(w)_0$ and w , and the reference path are plotted. Note that the subplots have been rotated clockwise by 90 degrees for formatting purpose. We observe that the output of the BP problem imitates the target driver's driving style to some extent and generates vehicle paths that approximately match the demonstrations with small deviations. Exact matching is not observed as the problem converges to some local minimum. As discussed earlier, test cases where obstacle centers are positioned higher than (in this case, to the right of) the reference path exhibit less improvement, and the reverse holds as well. A comparison of the reference path and the 10 demonstrations can be found in Appendix B.3.

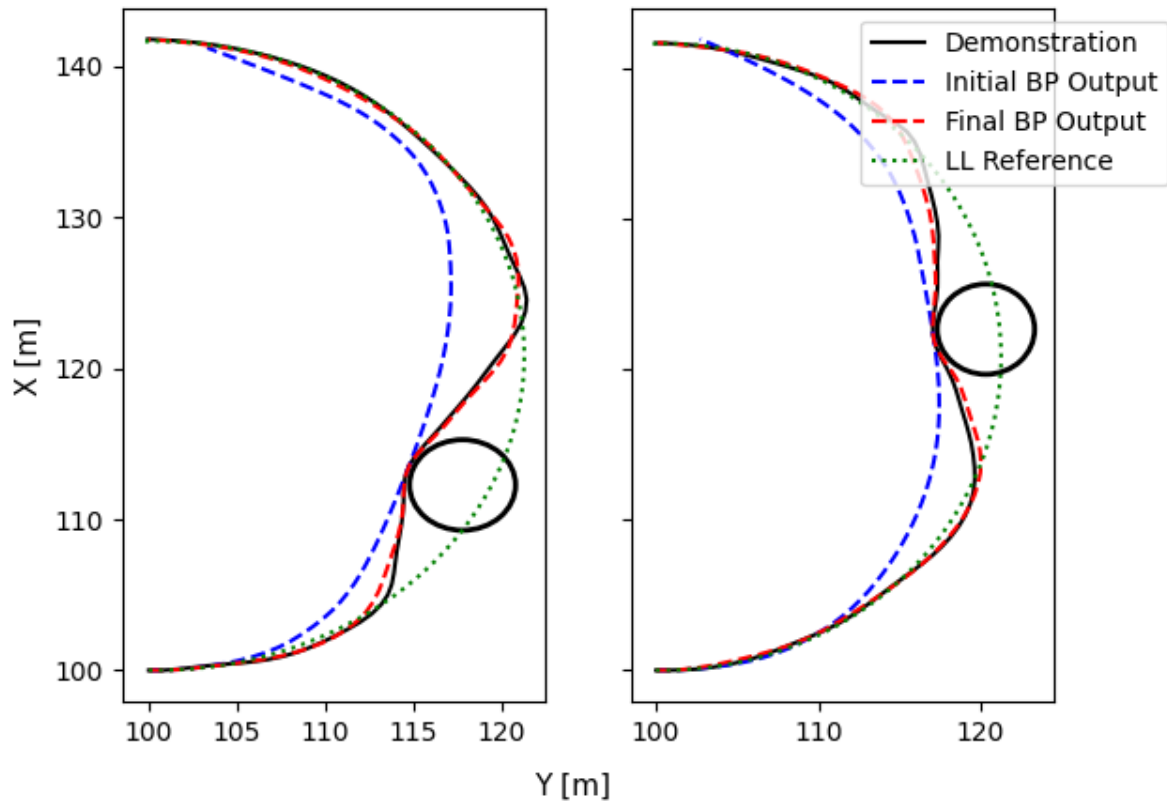


Figure 5.15: Personalized vehicle motion planning at a roundabout with obstacle using the first Demonstrations 9-10.

Chapter 6

Conclusion

In this work, we present the formulation and solution methods of a parameterized nonconvex trajectory optimization problem capable of expert behaviour mimicking, applied to a personalized [AV](#) controller. Constructed as a [BP](#) problem, the formulation consists of a vehicle motion planning layer and a driving style imitating layer. The vehicle motion planning layer minimizes a cost function parameterized by weights and produces personalized vehicle trajectories given the target user’s weight combination. The driving style imitating layer minimizes the difference between generated vehicle trajectories and user’s driving data, solving which a weight combination that characterizes a target driver’s driving style can be obtained.

To solve the vehicle motion planning subproblem, we propose an input-to-state reformation and the [SQP-TS](#) algorithm to solve trajectory optimization problems and apply this method to generate paths for an autonomous vehicle. Convergence is guaranteed for problems with input-affine inequalities. This method shows promise, as it converges for a larger variety of test cases in practice, including obstacle avoidance with complex reference paths. In addition, it requires significantly fewer iterations to converge than the traditional [SQP](#) solvers, thanks to the improved quality of local approximation by using trajectory sensitivities to reduce numerical errors in the states.

As a next step, we highlight that the convergence analysis applies only to a type of nonconvex optimization problem with input-affine inequality constraints. Therefore, we need to extend the proof to a more general case in the future. Also, we would like to explore the proposed algorithm’s compatibility with reference paths generated using existing path planners. Furthermore, while the solution discussed in this paper is based on an [SQP](#) formulation, trajectory sensitivities and their corresponding local approximation

with higher accuracy could also be applied to quasi-Newton or other trajectory optimization algorithms. We hope to explore the benefits of using trajectory sensitivities to improve numerical performance of these other methods as well.

To solve the driving style imitating subproblem, we adopt a projected subgradient update method with vanishing step sizes. We derive an expression of the **UL** subgradient (i.e., the subgradient of the **UL** problem objective with respect to the **UL** weights), which contains the **LL** subgradient (i.e., the subgradient of the **LL** solution with respect to the **UL** weights). We perform a sensitivity analysis of the **LL** optimization problem with input-to-state reformulation to solve for the **LL** subgradient. The proposed algorithm has been tested in simulation to perform the task of tracking a driver’s demonstration with curved or nonsmooth reference paths and even with obstacles. Simulation results show that we are able to accurately imitate a target driver’s driving style and reproduce very similar vehicle trajectories.

Here we highlight the limitation of using the proposed method to solve the driving style imitating problem. With vanishing step sizes, the change in the **UL** decision variables decreases as the number of iterations increases, leading to slower convergence. Moving on, we would like to implement a line search algorithm and use adaptive step length during projected subgradient update. Similarly, we may explore a trust-region-based approach for the **UL** problem using the **LL** subgradients to approximate $u^*(w)$. Also, we hope to derive a convergence proof of the projected subgradient update method when solving nonconvex, nonsmooth optimization problems.

Looking at the personalized **AV** control problem as a whole, we highlight two limitations of the method. First, the reference paths for the **LL** problem are generated given the starting and end points and the time it takes to travel between the two points. This information is extracted from a demonstration. In other words, the average velocity is somewhat "assumed" by the algorithm, instead of being imitated. Therefore, the personalized **AV** controller is not able to find a suitable average velocity on its own. Second, the expressiveness of the **LL** objective function depends on the cost metrics that we consider. Hence, if not enough cost metrics are included, then the imitated driving style may not be as representative of the actual driving style. Here, we exchange the accuracy of driving style imitation with interpretability of the cost function.

To address the first limitation, we may treat the speed of the vehicle as another parameter of driving style. However, this parameter will not be consistent for the same driver and will depend heavily on the driving scenarios. To handle the second limitation, various objective functions need to be analyzed and tested to evaluate the resulting controller performance.

Finally, more simulations may be helpful to better assess the performance of the proposed method. It would be interesting to investigate the generalizability of the method by obtaining the optimal weight from demonstrations of one driving scenario and testing it in a different scenario. In addition, we would like to compare our method with existing [AI](#)-based methods in terms of robustness, generalizability, and quality of driving style mimicking. This may also facilitate us in determining a suitable objective function for our method.

References

- [1] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145, 2017.
- [2] Alan H Barr. The einstein summation notation. *An Introduction to Physically Based Modeling (Course Notes 19)*, pages E, 1:57, 1991.
- [3] Chandrayee Basu, Qian Yang, David Hungerman, Mukesh Singhal, and Anca D Dragan. Do you want your autonomous car to drive like you? In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 417–425, 2017.
- [4] Paul T. Boggs and Jon W. Tolle. Sequential quadratic programming. *Acta Numerica*, 4:1–51, 1995.
- [5] Drew Bolduc, Longxiang Guo, and Yunyi Jia. Modeling and characterization of driving styles for adaptive cruise control in personalized autonomous vehicles. In *Dynamic Systems and Control Conference*, volume 58271, page V001T44A004. American Society of Mechanical Engineers, 2017.
- [6] Stephen Boyd, Lin Xiao, and Almir Mutapcic. Subgradient methods. *Lecture Notes of EE392o, Stanford University, Autumn Quarter, 2004(01)*, 2003.
- [7] Runqi Chai, Al Savvaris, Antonios Tsourdos, Senchun Chai, Runqi Chai, Al Savvaris, Antonios Tsourdos, and Senchun Chai. Overview of trajectory optimization techniques. *Design of Trajectory Optimization Approach for Space Maneuver Vehicle Skip Entry Problems*, pages 7–25, 2020.
- [8] Runqi Chai, Al Savvaris, Antonios Tsourdos, Senchun Chai, and Yuanqing Xia. A review of optimization techniques in spacecraft flight trajectory design. *Progress in Aerospace Sciences*, 109:100543, 2019.

- [9] Jieneng Chen, Jingye Chen, Ruiming Zhang, and Xiaobin Hu. Toward a brain-inspired system: Deep recurrent reinforcement learning for a simulated self-driving agent. *Frontiers in Neurobotics*, 13:40, 2019.
- [10] Shuping Chen, Huiyan Chen, and Dan Negrut. Implementation of mpc-based path tracking for autonomous vehicles considering three vehicle dynamics models with different fidelities. *Automotive Innovation*, 3(4):386–399, 2020.
- [11] Frank H Clarke. *Optimization and nonsmooth analysis*. SIAM, 1990.
- [12] Damek Davis, Dmitriy Drusvyatskiy, Sham Kakade, and Jason D Lee. Stochastic sub-gradient method converges on tame functions. *Foundations of Computational Mathematics*, 20(1):119–154, 2020.
- [13] Laura Eboli, Gabriella Mazzulla, and Giuseppe Pungillo. How drivers’ characteristics can affect driving style. *Transportation Research Procedia*, 27:945–952, 2017.
- [14] Yingying Feng and Xiaolong Yan. Support vector machine based lane-changing behavior recognition and lateral trajectory prediction. *Computational Intelligence and Neuroscience*, 2022(1):3632333, 2022.
- [15] Michael W Fisher. Computing safety margins of parameterized nonlinear systems for vulnerability assessment via trajectory sensitivities. *arXiv preprint arXiv:2501.07498*, 2025.
- [16] Michael W Fisher and Ian A Hiskens. Numerical computation of critical system recovery parameter values by trajectory sensitivity maximization. In *IEEE Conference on Decision and Control (CDC)*, pages 8000–8006, 2019.
- [17] Bingzhao Gao, Kunyang Cai, Ting Qu, Yunfeng Hu, and Hong Chen. Personalized adaptive cruise control based on online driving style recognition technology and model predictive control. *IEEE Transactions on Vehicular Technology*, 69(11):12482–12496, 2020.
- [18] David González, Joshué Pérez, Vicente Milanés, and Fawzi Nashashibi. A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 17(4):1135–1145, 2015.
- [19] Yanlei Gu, Yoriyoshi Hashimoto, Li-Ta Hsu, Miho Iryo-Asano, and Shunsuke Kamijo. Human-like motion planning model for driving in signalized intersections. *IATSS Research*, 41(3):129–139, 2017.

- [20] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024.
- [21] Peng Hang, Chen Lv, Yang Xing, Chao Huang, and Zhongxu Hu. Human-like decision making for autonomous driving: A noncooperative game theoretic approach. *IEEE Transactions on Intelligent Transportation Systems*, 22(4):2076–2087, 2020.
- [22] Jeffrey Hawke, Richard Shen, Corina Gurau, Siddharth Sharma, Daniele Reda, Nikolay Nikolov, Przemysław Mazur, Sean Micklethwaite, Nicolas Griffiths, Amar Shah, et al. Urban driving with conditional imitation learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 251–257. IEEE, 2020.
- [23] Ian A Hiskens and M Anantha Pai. Trajectory sensitivity analysis of hybrid systems. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 47(2):204–220, 2000.
- [24] M Hoedemaeker. *Driving behaviour with ACC and the acceptance by individual drivers*. IEEE, 2000.
- [25] Taylor A Howell, Simon Le Cleac’h, Sumeet Singh, Pete Florence, Zachary Manchester, and Vikas Sindhvani. Trajectory optimization with optimization-based dynamics. *IEEE Robotics and Automation Letters*, 7(3):6750–6757, 2022.
- [26] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.
- [27] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. In *2011 IEEE International Conference on Robotics and Automation*, pages 1478–1483. iee, 2011.
- [28] Yann Koeberle, Stefano Sabatini, Dzmitry Tsishkou, and Christophe Sabourin. Learning human like driving policies from real interactive driving scenes. 2022.
- [29] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099. IEEE, 2015.
- [30] Steven George Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2002.
- [31] Markus Kuderer, Shilpa Gulati, and Wolfram Burgard. Learning driving styles for autonomous vehicles from demonstration. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2641–2646. IEEE, 2015.

- [32] John M Lee and John M Lee. *Smooth manifolds*. Springer, 2012.
- [33] T-HS Li, Shih-Jie Chang, and Yi-Xiang Chen. Implementation of human-like driving skills by autonomous fuzzy behavior control on an fpga-based car-like mobile robot. *IEEE Transactions on Industrial Electronics*, 50(5):867–880, 2003.
- [34] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009.
- [35] Jiali Ling, Jialong Li, Kenji Tei, and Shinichi Honiden. Towards personalized autonomous driving: An emotion preference style adaptation framework. In *2021 IEEE International Conference on Agents (ICA)*, pages 47–52. IEEE, 2021.
- [36] Zheng Ma and Yiqi Zhang. Investigating the effects of automated driving styles and driver’s driving styles on driver trust, acceptance, and take over behaviors. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 64, pages 2001–2005. Sage Publications Sage CA: Los Angeles, CA, 2020.
- [37] Danylo Malyuta, Yue Yu, Purnanand Elango, and Behçet Açıkmeşe. Advances in trajectory optimization for space vehicle control. *Annual Reviews in Control*, 52:282–315, 2021.
- [38] Soroush Nasiriany, Huihan Liu, and Yuke Zhu. Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7477–7484, 2022.
- [39] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer, 1999.
- [40] Daniel Omeiza, Sule Anjomshoae, Helena Webb, Marina Jirotko, and Lars Kunze. From spoken thoughts to automated driving commentary: Predicting and explaining intelligent vehicles’ actions. In *2022 IEEE Intelligent Vehicles Symposium (IV)*, pages 1040–1047. IEEE, 2022.
- [41] Benjamin J Patz, Yiannis Papelis, Remo Pillat, Gary Stein, and Don Harper. A practical approach to robotic design for the darpa urban challenge. *Journal of Field Robotics*, 25(8):528–566, 2008.
- [42] Alice Plebe, Henrik Svensson, Sara Mahmoud, and Mauro Da Lio. Human-inspired autonomous driving: A survey. *Cognitive Systems Research*, 83:101169, 2024.

- [43] Anil V Rao. Trajectory optimization: a survey. *Optimization and Optimal Control in Automotive Systems*, pages 3–21, 2014.
- [44] Stephen M Robinson. Perturbed kuhn-tucker points and rates of convergence for a class of nonlinear-programming algorithms. *Mathematical Programming*, 7:1–16, 1974.
- [45] Ugo Rosolia, Stijn De Bruyne, and Andrew G Alleyne. Autonomous vehicle control: A nonconvex approach for obstacle avoidance. *IEEE Transactions on Control Systems Technology*, 25(2):469–484, 2016.
- [46] Saumya Kumar Saksena, B Navaneethkrishnan, Sinchana Hegde, Pragadeesh Raja, and Ravi M Vishwanath. Towards behavioural cloning for autonomous driving. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 560–567. IEEE, 2019.
- [47] Mariah L Schrum, Emily Sumner, Matthew C Gombolay, and Andrew Best. Maveric: A data-driven approach to personalized autonomous driving. *IEEE Transactions on Robotics*, 40:1952–1965, 2024.
- [48] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.
- [49] Max Schwenger, Muzaffer Ay, Thomas Bergs, and Dirk Abel. Review on model predictive control: An engineering perspective. *The International Journal of Advanced Manufacturing Technology*, 117(5):1327–1349, 2021.
- [50] Shobit Sharma, Girma Tewolde, and Jaerock Kwon. Behavioral cloning for lateral motion control of autonomous vehicles using deep learning. In *2018 IEEE International Conference on Electro/Information Technology (EIT)*, pages 0228–0233. IEEE, 2018.
- [51] Mingfei Sun and Xiaojuan Ma. Adversarial imitation learning from incomplete demonstrations. *arXiv preprint arXiv:1905.12310*, 2019.
- [52] Xu Sun, Jingpeng Li, Pinyan Tang, Siyuan Zhou, Xiangjun Peng, Hao Nan Li, and Qingfeng Wang. Exploring personalised autonomous vehicles to influence user trust. *Cognitive Computation*, 12(6):1170–1186, 2020.

- [53] Lars Svensson and Jenny Eriksson. Tuning for ride quality in autonomous vehicle: Application to linear quadratic path planning algorithm, 2015.
- [54] Orit Taubman-Ben-Ari and Vera Skvirsky. The multidimensional driving style inventory a decade later: Review of the literature and re-evaluation of the scale. *Accident Analysis & Prevention*, 93:179–188, 2016.
- [55] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.
- [56] Hong Wang, Yanjun Huang, Amir Khajepour, Yubiao Zhang, Yadollah Rasekhipour, and Dongpu Cao. Crash mitigation in motion planning for autonomous vehicles. *IEEE transactions on intelligent transportation systems*, 20(9):3313–3323, 2019.
- [57] Zhepei Wang, Xin Zhou, Chao Xu, and Fei Gao. Geometrically constrained trajectory optimization for multicopters. *IEEE Transactions on Robotics*, 38(5):3259–3278, 2022.
- [58] Zheng Wu, Liting Sun, Wei Zhan, Chenyu Yang, and Masayoshi Tomizuka. Efficient sampling-based maximum entropy inverse reinforcement learning with application to autonomous driving. *IEEE Robotics and Automation Letters*, 5(4):5355–5362, 2020.
- [59] Donghao Xu, Zhezhong Ding, Xu He, Huijing Zhao, Mathieu Moze, François Aioun, and Franck Guillemand. Learning from naturalistic driving data for human-like autonomous highway driving. *IEEE Transactions on Intelligent Transportation Systems*, 22(12):7341–7354, 2020.
- [60] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020.
- [61] Nidzamuddin Md Yusof, Juffrizal Karjanto, Jacques Terken, Frank Delbressine, Muhammad Zahir Hassan, and Matthias Rauterberg. The exploration of autonomous vehicle driving styles: Preferred longitudinal, lateral, and vertical accelerations. In *Proceedings of the 8th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pages 245–252, 2016.
- [62] Andrea Zanelli, Alexander Domahidi, Juan Jerez, and Manfred Morari. Forces nlp: An efficient implementation of interior-point methods for multistage nonlinear nonconvex programs. *International Journal of Control*, 93(1):13–29, 2020.

- [63] Zhiqiang Zuo, Xu Yang, Zheng Li, Yijing Wang, Qiaoni Han, Li Wang, and Xiaoyuan Luo. Mpc-based cooperative control strategy of path planning and trajectory tracking for intelligent vehicles. *IEEE Transactions on Intelligent Vehicles*, 6(3):513–522, 2020.

APPENDICES

Appendix A

Subgradient Computation

A.1 Analysis of LL Subgradient Computation

Consider the matrix equality in (4.1). Denote the matrix on the left-hand side by A . By performing row and column manipulation of (4.1), A can be rewritten as

$$\begin{bmatrix} P_1 & 0 & 0 & -I \\ 0 & P_2 + \frac{d}{d\mathbf{u}}(dG_u^T \lambda) + \frac{d}{d\mathbf{u}}(dH_u^T \nu) & dH_u^T & dG_u^T \\ 0 & D(\nu)dH_u & D(H(u)) & 0 \\ -I & dG_u & 0 & 0 \end{bmatrix}.$$

Now, let \mathcal{I} be the set where $\nu^* > 0$. Under the strict complementarity assumption, the corresponding inequality constraints $D(H(u))_{\mathcal{I}}$ are evaluated to zero. By looking at these active inequality constraints only, the matrix now becomes:

$$\begin{bmatrix} P_1 & 0 & 0 & -I \\ 0 & P_2 + \frac{d}{d\mathbf{u}}(dG_u^T \lambda) + \frac{d}{d\mathbf{u}}(dH_u^T \nu) & (dH_u^T)_{\mathcal{I}} & dG_u^T \\ 0 & (D(\nu)dH_u)_{\mathcal{I}} & 0 & 0 \\ -I & dG_u & 0 & 0 \end{bmatrix},$$

which is still a square matrix. Observe that the first and fourth columns have full column rank, and that the first and fourth rows have full row rank, thanks to the identity matrix. Under the linear independence constraint qualification, the third row has full row rank given dG_u having full row rank, and the third column has full column rank given $(dH_u^T)_{\mathcal{I}}$ having full column rank. In addition, P_2 is a positive definite matrix of full rank; adding

to a full-rank matrix will generally also result in a matrix of full rank. Thus, the second row / column has full row /column rank. Putting everything together, the matrix has full rank and hence a solution exists.

When the strict complimentary slackness assumption does not hold, we have the following:

$$[D(\nu)dH_u]_i = [D(H(u))]_i = 0$$

for some $i \in \{1, \dots, p\}$. Since the corresponding entries on the right-hand side of the matrix equalities all equal to zero, there still exists a solution, but the solution is not unique. This corresponds to the case where F is not smooth and $\frac{dF}{dw}$ is an element of the subdifferential [1].

A.2 LL Subgradient without Input-to-State Map

Recall the bilevel program without state-to-input mapping

$$\begin{aligned} \min_w \quad & J_U(\mathbf{y}(\mathbf{u}_*, w_1, w_2)) \\ \text{s.t.} \quad & \mathbf{u}^* = \arg \min_u \frac{1}{2}(\mathbf{x} - \mathbf{x}_r)^T P_1(w_1)(\mathbf{x} - \mathbf{x}_r) + \frac{1}{2}\mathbf{u}^T P_2(w_2)\mathbf{u} \\ & \text{s.t.} \quad x^{k+1} = g(x^k, u^k) \quad \forall k \in \{0, \dots, T-1\}, \\ & \quad h(x^k, u^k) \leq 0 \quad \forall k \in \{0, \dots, T-1\}, \\ & \quad y^k = Cx^k \quad \forall k \in \{0, \dots, T-1\}, \\ & \quad \mathbf{x} = [(x^0)^\top, \dots, (x^{T-1})^\top]^\top, \quad \mathbf{u} = [(u^0)^\top, \dots, (u^{T-1})^\top]^\top, \\ & \quad \mathbf{y} = [(y^0)^\top, \dots, (y^{T-1})^\top]^\top. \end{aligned}$$

The **UL** subgradient can be computed, in this case, as

$$\left(\frac{dJ_U}{dw} \right)_{Jac} = \frac{\partial J_U}{\partial \mathbf{y}} \bar{C} \left(\frac{\partial \mathbf{x}}{\partial \mathbf{u}_*} \frac{dF}{dw} \right)_{Jac}, \quad (\text{A.1})$$

where the second term on the right-hand side is

$$\left(\frac{\partial \mathbf{x}}{\partial \mathbf{u}}\right)_{Jac} = \begin{bmatrix} \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \cdots & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} \\ \begin{pmatrix} \frac{\partial x_1^1}{\partial u_1^0} \\ \vdots \\ \frac{\partial x_6^1}{\partial u_1^0} \end{pmatrix} & \begin{pmatrix} \frac{\partial x_1^1}{\partial u_2^0} \\ \vdots \\ \frac{\partial x_6^1}{\partial u_2^0} \end{pmatrix} & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \cdots & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} \\ \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \begin{pmatrix} \frac{\partial x_1^2}{\partial u_1^1} \\ \vdots \\ \frac{\partial x_6^2}{\partial u_1^1} \end{pmatrix} & \begin{pmatrix} \frac{\partial x_1^2}{\partial u_2^1} \\ \vdots \\ \frac{\partial x_6^2}{\partial u_2^1} \end{pmatrix} & \cdots & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} \\ \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \cdots & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \mathbf{0}_{6 \times 1} & \cdots & \mathbf{0}_{6 \times 1} & \begin{pmatrix} \frac{\partial x_1^T}{\partial u_1^{T-1}} \\ \vdots \\ \frac{\partial x_6^T}{\partial u_1^{T-1}} \end{pmatrix} & \begin{pmatrix} \frac{\partial x_1^T}{\partial u_2^{T-1}} \\ \vdots \\ \frac{\partial x_6^T}{\partial u_2^{T-1}} \end{pmatrix} \end{bmatrix},$$

which is a matrix that is almost block-diagonal (off by 6 rows) and contains many zeros.

To obtain the third term on the right-hand side of (A.1), we look closer at the lower-level problem, whose Lagrangian is

$$L = \frac{1}{2}(\mathbf{x} - \mathbf{x}_r)^\top P_1(\mathbf{x} - \mathbf{x}_r) + \frac{1}{2}\mathbf{u}^\top P_2\mathbf{u} + \lambda^\top(\bar{g}(\mathbf{x}, \mathbf{u}) - \mathbf{x}) + \nu^\top \bar{h}(\mathbf{x}, \mathbf{u}) - \mathbf{1}_{\nu \geq 0},$$

where \bar{g} is the vector of functions of all vehicle dynamics on the prediction horizon. Taking the gradient of this Lagrangian, we get the following optimality condition:

$$\nabla L = 0 = f(\mathbf{x}, \mathbf{u}, w_1, w_2, \lambda, \nu).$$

Writing out the explicit expression of ∇L , we get

$$\begin{bmatrix} P_1(\mathbf{x} - \mathbf{x}_r) + \frac{d}{d\mathbf{x}}(\bar{g})\lambda^\top \\ P_2\mathbf{u} + \frac{d}{d\mathbf{u}}(\bar{g})\lambda^\top + \frac{d}{d\mathbf{u}}(\bar{h})\nu^\top \\ \bar{g}(\mathbf{x}, \mathbf{u}) - \mathbf{x} \\ \text{Diag}(\nu)\bar{h}(\mathbf{x}, \mathbf{u}) \end{bmatrix} = \begin{bmatrix} P_1(\mathbf{x} - \mathbf{x}_r) + \frac{d}{d\mathbf{x}}(\bar{g})\lambda^\top \\ P_2\mathbf{u} + \frac{d}{d\mathbf{u}}(\bar{g})\lambda^\top + \frac{d}{d\mathbf{u}}(\bar{h})\nu^\top \\ \bar{g}(\mathbf{x}, \mathbf{u}) - \mathbf{x} \\ \text{Diag}(\bar{h})\nu \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Use the Implicit Function Theorem and take the differentials of $\nabla L = 0$, then rearrange to get a matrix equality:

$$\begin{bmatrix} P_1 + \frac{\partial}{\partial \mathbf{x}} \left(\frac{\partial \bar{g}}{\partial \mathbf{x}} \lambda^\top \right) & \frac{\partial}{\partial \mathbf{u}} \left(\frac{\partial \bar{g}}{\partial \mathbf{x}} \lambda^\top \right) & -I & 0 \\ \frac{\partial}{\partial \mathbf{x}} \left(\frac{\partial \bar{g}}{\partial \mathbf{u}} \lambda^\top + \frac{\partial \bar{h}}{\partial \mathbf{u}} \lambda^\top \right) & P_2 + \frac{\partial}{\partial \mathbf{u}} \left(\frac{\partial \bar{g}}{\partial \mathbf{u}} \lambda^\top + \frac{\partial \bar{h}}{\partial \mathbf{u}} \lambda^\top \right) & \frac{\partial \bar{g}}{\partial \mathbf{u}}^\top & \frac{\partial \bar{h}}{\partial \mathbf{u}}^\top \\ \frac{\partial \bar{g}}{\partial \mathbf{x}} - I & \frac{\partial \bar{g}}{\partial \mathbf{u}} & 0 & 0 \\ 0 & \text{Diag}(\nu) \frac{\partial \bar{h}}{\partial \mathbf{u}}^\top & 0 & \text{Diag}(\bar{h}) \end{bmatrix} \begin{bmatrix} \frac{\partial \mathbf{x}}{\partial w} \\ \frac{\partial \mathbf{u}}{\partial w} \\ \frac{\partial \lambda}{\partial w} \\ \frac{\partial \nu}{\partial w} \end{bmatrix} = \begin{bmatrix} -\frac{\partial P_1(\mathbf{x}-\mathbf{x}_r)}{\partial w_1} & 0 \\ 0 & -\frac{\partial P_2 \mathbf{u}}{\partial w_2} \\ 0 & 0 \\ 0 & 0 \end{bmatrix},$$

solving which the matrix $\left(\frac{\partial \mathbf{u}}{\partial w} \right)_{Jac}$ can be obtained.

Appendix B

Additional Information for Simulation

B.1 Vehicle Parameters

Symbol	Definition	Unit
$C_{\alpha f}$	front tire cornering stiffness	N/rad
$C_{\alpha r}$	rear tire cornering stiffness	N/rad
$C_{x f}$	front tire longitudinal stiffness	N
$C_{x r}$	rear tire longitudinal stiffness	N
s_f	front tire slip ratio	/
s_r	rear tire slip ratio	/
a	distance of front tire to center of mass	m
b	distance of rear tire to center of mass	m
m	vehicle mass	kg
I_z	vehicle yaw inertia	$kg \cdot m^2$

Table B.1: Definition of Vehicle Parameters

B.2 Sampled Weights and Obstacle Positions

Table B.2: Sampled weights for tracking of a curved path

Demo	w_{r0}	w_{r1}	w_{r2}	w_{r3}
1	1.037587331	1.037587331	10.22220496	0.968977362
2	1.03318123	1.03318123	10.15507826	0.953066739
3	1.014639061	1.014639061	10.47586751	0.954230447
4	0.952933699	0.952933699	10.24145105	0.954133463
5	1.011351539	1.011351539	10.08889766	0.994294925
6	0.962863072	0.962863072	9.738972023	1.024629146
7	1.045410779	1.045410779	9.770136574	0.984552125
8	0.966018091	0.966018091	10.14758925	1.008989146
9	0.961727414	0.961727414	9.585629082	1.043310596
10	0.985046126	0.985046126	9.908178105	1.016600511

Table B.3: Sampled weights for tracking of a complex path

Demo	w_{r0}	w_{r1}	w_{r2}	w_{r3}
1	1.040434748	1.040434748	9.971409084	0.991543361
2	1.036816312	1.036816312	9.809844506	1.044244401
3	0.98898905	0.98898905	9.99551078	0.984036836
4	1.012924019	1.012924019	9.866789362	0.994879884
5	0.975641298	0.975641298	10.20558646	1.005074836
6	0.994562093	0.994562093	9.537985332	0.970957257
7	0.986755748	0.986755748	9.801108117	1.049241417
8	0.999079811	0.999079811	9.739465463	1.030233092
9	0.975610884	0.975610884	9.802349451	0.987546971
10	0.990698942	0.990698942	10.3781268	1.003836749

Table B.4: Sampled weights and obstacle positions for tracking of a complex path

Demo	w_{r0}	w_{r1}	w_{r2}	w_{r3}	x_{obs}	y_{obs}
1	1.000789411	1.000789411	0.993694892	0.993694892	118.9418579	110.0601006
2	1.045627227	1.045627227	1.044219958	1.044219958	116.1170886	110.4491545
3	1.03667918	1.03667918	0.97708309	0.97708309	118.9386452	123.0928298
4	0.983410011	0.983410011	1.047936497	1.047936497	123.3676281	116.5037207
5	1.034797346	1.034797346	0.953001597	0.953001597	118.7531941	116.7107152
6	1.026750708	1.026750708	1.026718564	1.026718564	120.5211826	129.5824678
7	1.031933936	1.031933936	1.02473191	1.02473191	117.0533157	111.2989963
8	1.031933936	1.031933936	1.02473191	1.02473191	120.6635841	111.5480478
9	0.994702571	0.994702571	1.007650947	1.007650947	117.7619122	112.2699468
10	0.969776018	0.969776018	1.023144894	1.023144894	120.3195948	122.6764469

B.3 Additional Figures

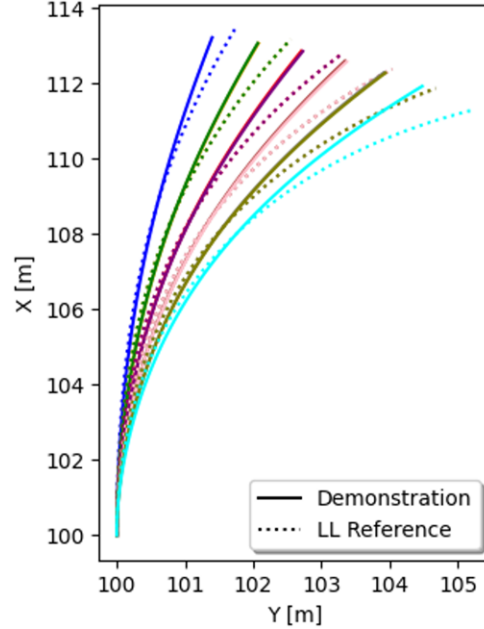


Figure B.1: Reference paths and simulated driving data following a curved path using 10 demonstrations.

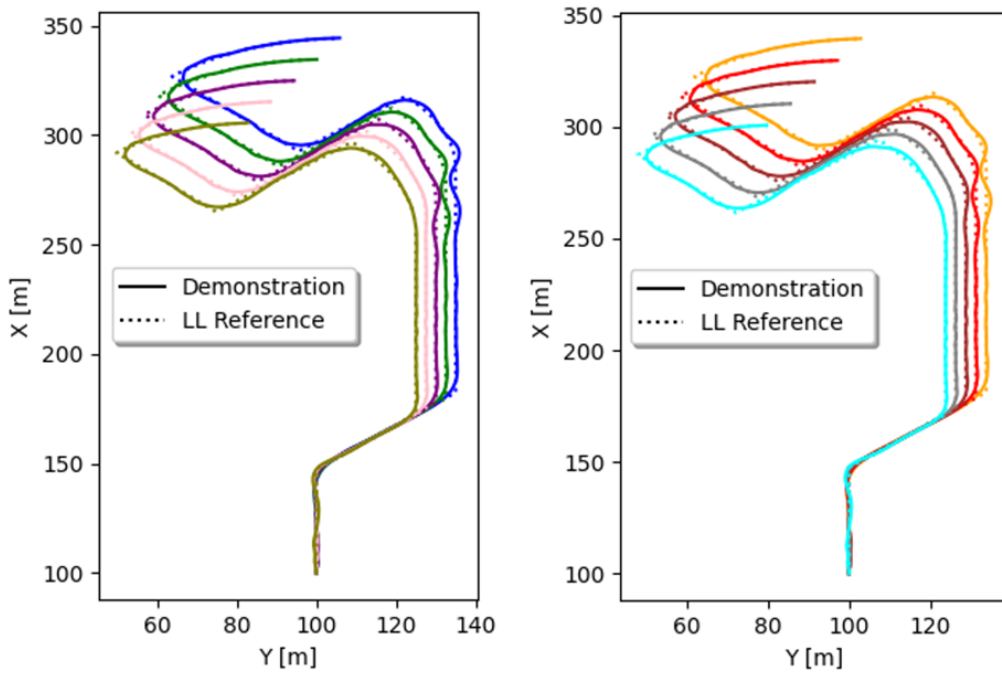


Figure B.2: Reference paths and simulated driving data following a complex path using 10 demonstrations.

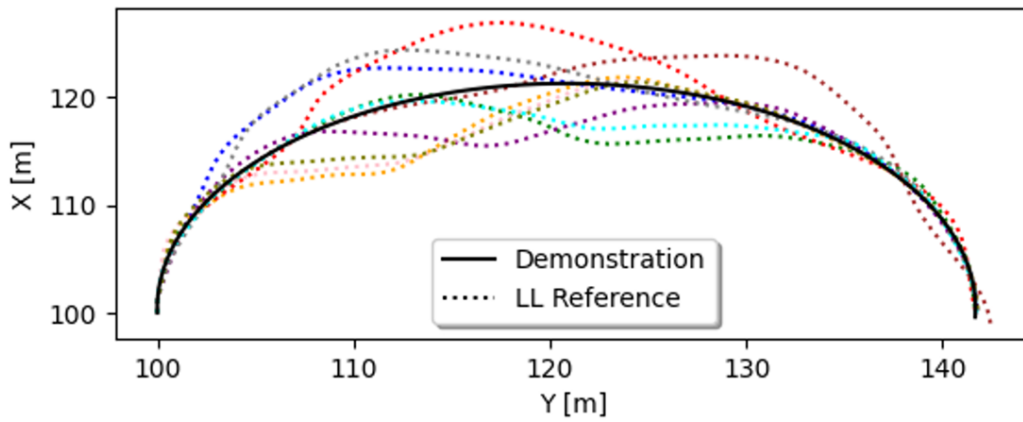


Figure B.3: Reference paths and simulated driving data at a roundabout with obstacle using 10 demonstrations.