Motion Planning for Autonomous Vehicles in Urban Scenarios: A Sequential Optimization Approach

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Wenda Xu

B.S., Automation, Tsinghua University

M.S., Machine Intelligence, Peking University

Carnegie Mellon University

Pittsburgh, PA

May 2021

Acknowledgments

I would like to express my sincere gratitude to my advisor John M. Dolan for the continuous support of my study, for his patience and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my study. I would also like to thank Maxim Likhachev, Stephen Smith, and Anthony Rowe for their thoughtful advice and diligent service as members of my dissertation committee. I would like to thank GM Research & Development and Delphi Electronics & Safety for supporting my PhD research.

Thanks to Jarrod Snider and Junqing Wei for years of advice, discussions, and friendship. Thanks to Qian Wang for insightful discussions on trajectory optimization. Thanks to the entire ADCRL team of CMU for their efforts and good discussions. My work during my Ph.D study required numerous field experiments on an autonomous vehicle. It would not have been possible without their help.

Finally, I would like to thank to my wife and my son for always being there and cheering me up and standing by me through the good times and bad. Thanks to my parents for always being supporting me and encouraging me with their best wishes.

Abstract

Motion planning is essential for an autonomous vehicle to perform safe and human-like driving behaviors, especially in highly dynamic scenarios such as dense urban and highway environments. The motion planning problem is challenging in that it needs to handle static and dynamic obstacles and obey kinematic and dynamic constraints as well as traffic rules. In this work, we propose an efficient hierarchical motion planning approach based on path-speed decomposition. We first plan a path to avoid static obstacles, then generate a speed profile along the path to interact with dynamic obstacles.

The first sub-problem is path planning with static obstacles. It can be viewed as non-convex constrained nonlinear optimization, which requires a good enough initial guess to start and is often sensitive to algorithm parameters. We formulate it as convex spline optimization. The convexity of the formulated problem makes it able to be solved fast and reliably, while guaranteeing a global optimum. We then reorganize the constrained spline optimization into a recurrent formulation, which further reduces the computational time to be linear in the optimization horizon size.

The second sub-problem is the minimum-time speed planning problem over a fixed path with dynamic obstacle constraints and point-wise speed and acceleration constraints. The contributions of our speed planning method are three-fold. First, we formulate the speed planning as an iterative convex optimization problem based on space discretization. Our formulation allows imposing dynamic obstacle

constraints and point-wise speed and acceleration constraints simultaneously. Second, we propose a modified vertical cell decomposition method to handle dynamic obstacles. It divides the freespace into channels, where each channel represents a homotopy of free paths and defines convex constraints for dynamic obstacles. Third, we demonstrate significant improvement over previous work on speed planning for typical driving scenarios such as following, merging, and crossing.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

According to the United State Census Bureau [1], there are around 10 million accidents in the United States every year. Most of them are caused by human distraction or operation error. Autonomous vehicles have great potential to improve the safety and performance of the transportation system. They can also free people from the task of driving, which could save commuters considerable time daily. To achieve this objective without affecting existing human drivers on the road, autonomous vehicles need to have human-acceptable driving performance. The vehicle's planner also needs to meet strict real-time requirements to react fast enough in emergency situations. In summary, it is important, but difficult, to develop a practical high-performance realtime motion planner for on-road driving.

1.2 Challenges

Mathematically, the motion planning problem is a high-dimensional non-convex optimization problem that is subject to static and dynamic obstacle constraints, kinematic constraints (e.g. curvature bounds), dynamic constraints (e.g. speed limit, acceleration limit, jerk limit), a nonlinear time-varying dynamic system model, traffic rules, smoothness requirements, and most importantly realtime requirements. Generating a feasible safe trajectory in realtime with proper assumptions is the biggest challenge in motion planning for autonomous vehicles. Traditional search-based motion planning methods suffer the curse of dimensionality, especially in spatiotemporal state space. The resulting paths from search-based methods are also not smooth enough for the vehicle to follow. Optimization-based motion planning methods don't ensure finding a feasible solution in a non-convex feasible region, and don't ensure converging to a global optimum for a non-convex objective function. Therefore, the worst-case performance of the optimization-based methods is not guaranteed. In this work, a novel motion planning approach is designed. By using the advantages of both search-based and optimization-based methods, the motion planning problem is solved efficiently.

1.3 Planning Framework

In this work, a hierarchical motion planning framework is proposed. As shown in Fig. 1.1, we decompose the motion planning problem into two sub-tasks. The first sub-task is to find a collision-free path among static obstacles. The second sub-task is to generate a time-optimal speed profile on the given path while keeping a safe distance from dynamic obstacles. The planner framework uses the advantage of

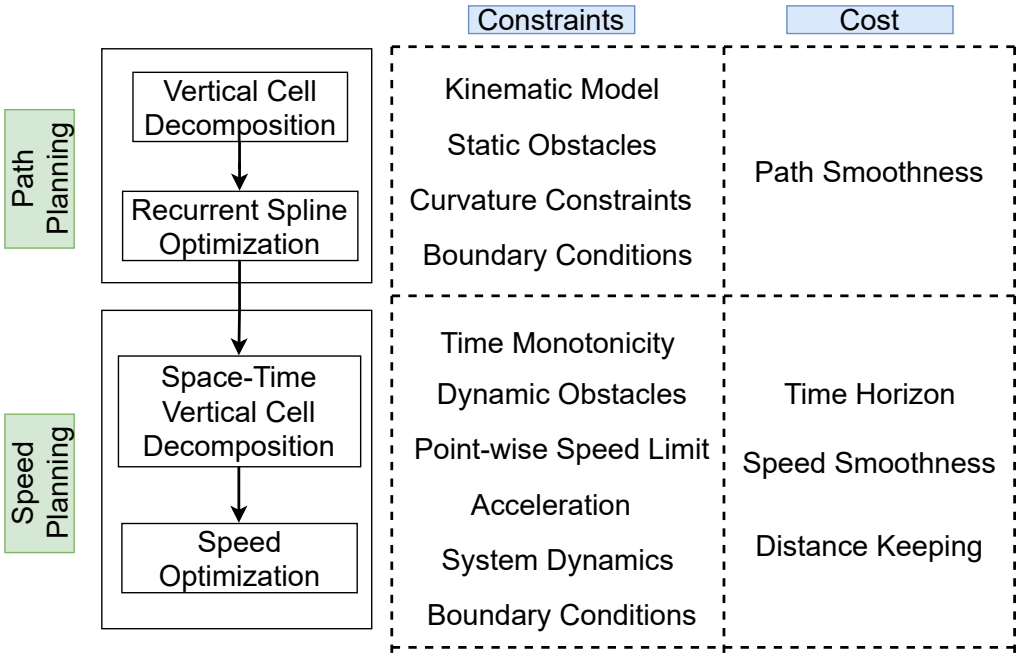


Figure 1.1: Planning framework.

both search-based and optimization-based methods. It consists of several layers.

First, we use a vertical cell decomposition method to handle static obstacles, where the free-space is partitioned into a set of cells. Each cell is either a trapezoid or a triangle. The corridor is built by a list of connected convex cells.

Second, given the path corridor, a convex path optimization is applied to obtain a collision-free smooth path that also obeys the kinematic model. Based on Differential Flatness theory, the path optimization is formulated as a constrained convex optimization problem, which guarantees a global optimum. The curvature constraints are handled by an iterative optimization approach.

Third, we propose a space-time vertical cell decomposition method to handle dynamic obstacles. Compared to the normal vertical cell decomposition method, additional constraints need to be considered. Any feasible solution should be no

faster than the minimum-time solution. Moreover, time should be monotonically increasing.

Fourth, given the coarse collision-free speed profile and the S-T corridor, a novel speed optimizer is proposed. Compared to previous methods, it can handle constraints on the station dimension such as velocity limit and stop line and constraints on the time dimension such as moving obstacles simultaneously based on iterative quadratic programming (QP). It can handle cases such as yielding to other traffic as well as merging in front of other vehicles.

1.4 Contributions

This work proposes a sequential optimization approach for autonomous urban driving. The planner can achieve the goal of simultaneously generating a smooth path and speed profile and satisfying kinematic and dynamic constraints in realtime. It enables the vehicles to avoid static obstacles and interact with dynamic obstacles such as following a leading car, dealing with cut-in cars, yielding to oncoming traffic, finishing lane change in tight spaces, interacting with pedestrians, and merging onto the highway. It is sufficiently computationally efficient to be used in realtime applications. The planner framework takes advantage of both search-based and optimization-based methods. Key contributions are summarized as follows.

- We convert path planning into a convex quadratic program, which enables a reliably global optimal trajectory.
- Our path optimization approach allows large step size thanks to the linear system model based on differential flatness theory. It also does not require an initial feasible solution, allowing an infeasible start.

- By reformulating the constrained spline optimization into optimal control form, we further reduce the computational time to be linear in the optimization horizon size.

- We formulate the speed planning as an iterative convex optimization problem based on space discretization. Our formulation allows imposing dynamic obstacle constraints and point-wise speed and acceleration constraints simultaneously.

- We propose a modified vertical cell decomposition method to handle dynamic obstacles. It divides the freespace into channels, where each channel represents a homotopy of free paths and defines convex constraints for dynamic obstacles.

- We demonstrate significant improvement over previous work on speed planning for typical driving scenarios such as following, merging, and crossing.

1.5 Outline

The rest of the dissertation is organized as follows.

- Chapter 2 discusses related work, including search-based and optimization-based planning methods, and speed planning methods.

- Chapter 3 presents a path planning approach based on recurrent spline optimization.

- Chapter 4 presents a speed planning approach to generate speed profiles along a fixed path via iterative optimization.

- Chapter 5 summarizes the dissertation and discusses possible future directions.

Chapter 2

Literature Review

2.1 Motion Planning in Unstructured and Structured Environments

Planning for autonomous vehicles can be categorized into two classes in terms of planning horizon: mission-level path planning and short-term motion planning [2, 3]. Mission-level path planning usually finds the shortest path to achieve mission goals within the road network [4, 5]. Short-term motion planning takes the output of mission planning as a reference path, generates a feasible trajectory for the vehicle to follow, and interacts with surrounding dynamic and static obstacles [6]. In this dissertation, we focus on short-term motion planning, which generates a trajectory whose length is usually within hundreds of meters, but varies depending on the speed of the vehicle.

Motion planning for autonomous vehicles can be divided into two types in terms of different environments: unstructured [7] and structured [2]. A typical example of an unstructured environment is off-road terrain, where usually the layout

of the environment is unknown or partially known beforehand. To solve off-road terrain navigation, the solution involves estimation of the traversability of the terrain and reasoning about the terrain interaction. The goal of a planner in off-road terrain is to traverse the terrain given a traversability map from the perception system of the robot. There are fewer constraints from the environment for planning given the absence structure in the environment. Compared to structured environments such as urban streets, there are also fewer moving objects that need to be handled by the planner. Another example of an unstructured environment is a parking lot [8], where an autonomous vehicle needs to reach a specific pose. The common characteristic for planning on off-road terrain and in parking lots is that the robot itself is moving at low speed and there are few moving obstacles.

In structured environments such as urban roads and highways, the goals of the planner include selecting and driving in a desired lane, distance keeping to surrounding vehicles, and avoiding static and dynamic obstacles [6]. It is notable that usually the road connectivity and the road shape are known to the planner, and the center line of the desired lane is chosen as a reference path. These environmental constraints help the planner reduce the search space, but also introduce challenges in that the robot needs to obey the environmental constraints and vehicle dynamic constraints at the same time [9]. In addition, the robot also needs to cooperate with other vehicles while obeying traffic rules [10]. Challenging scenarios include lane change, ramp merging, four-way stops, etc., where the robot needs not only to generate a safe trajectory but also to recognize other vehicles' intentions and interact with them [11]. These problems are extremely difficult to solve compared to the traditional collision-free type of motion planning.

2.2 Search-based Path Planning

Motion planning techniques can be classified into two groups according to the continuity of the state space: search-based planning [12, 13] and optimization-based planning [14, 15, 16]. In search-based planning, the state space is discretized into a grid or lattice. In optimization-based planning, the state space is continuous. Although search-based planning may lose accuracy due to discretization, it guarantees a global optimum with respect to its resolution. Optimization-based planning often falls into a local optimum because the planning problem is generally not convex and the global optimum is hard to find through optimization techniques. For search-based planning methods, there are two key steps: trajectory generation between two nodes and search within the graph. The trajectory generation step generates trajectories connecting nodes based on the differential constraints of the robot. In the search step, it finds a solution from the start state to the goal based on heuristics or random sampling.

2.2.1 Search Algorithm

There are two types of search algorithms depending on the expansion strategy: search on a deterministic graph and randomized sampling. The single-source shortest path (SSSP) on a graph is a well-studied problem. For a general weighted graph with V vertices and E edges, the single source shortest path problem can be solved using the Bellman-Ford Algorithm in $\mathcal{O}(VE)$ time [17, 18]. The Bellman-Ford Algorithm can handle graphs with negative edge weights. If the graph doesn't have negative weights, Dijkstra's algorithm can find the shortest distance from the source to all other vertices in $\mathcal{O}(E + V \log V)$ time [19]. For a Directed Acyclic Graph (DAG), the SSSP problem can be solved in $\mathcal{O}(V + E)$ time using Topological

Sorting [20]. For the single source single destination problem, heuristic-based methods (e.g. A* [21] and D* [22]) can be used to further accelerate Dijkstra's algorithm. For heuristic-based algorithms, a good estimate of cost from any vertex to the goal is essential. Otherwise, it may not find the optimal solution or in the worst case it has to visit all vertices to get the optimal solution. The planning problem for autonomous driving is complicated; especially when there are dynamic obstacles, it is very difficult to find an appropriate heuristic that is suitable for all scenarios. Some incremental search algorithms (e.g. D* Lite [23], anytime D* [24, 25]), are widely used in robotics navigation to save replanning time when the environment changes. However, they only work for typical planning problems with a fixed goal, while in on-road scenarios the short-term goal is changing all the time.

Search on a deterministic graph suffers the curse of dimensionality. Sampling-based methods are developed to efficiently search in high-dimensional spaces. Probabilistic roadmap (PRM) [26] first constructs a random graph (roadmap) through random samples from the configurations space of the robot. The final path is obtained by search on the graph. [27] extends the standard PRM to the dynamic environment. It first builds a roadmap that only considers static obstacles, then generates a near-time-optimal trajectory that avoids dynamic obstacles. Rapidly-exploring Random Tree (RRT) [13] randomly builds a tree in the search space until it reaches the goal. Both PRM and RRT are probabilistically complete, but an optimal solution is not guaranteed. The results are often not smooth enough for the car to execute.

2.2.2 Trajectory Generation

There are two types of trajectory generation methods: forward kinematics and inverse kinematics. In forward kinematics, the initial state of the robot and control input are given, and the final state can be obtained through integrating the control input. Typical numerical integration methods include the Euler, Simpson, and Runge-Kutta methods. In inverse kinematics, the initial and final state of the robot are given, and the control input is to be determined. It usually requires solving a system of algebraic equations if the trajectory can be represented by polynomials [28] or a trajectory optimization problem for general nonlinear systems [29, 30].

Simultaneously handling path and curvature constraints efficiently is a challenging problem. [31] uses curvature polynomials to ensure smooth curvature. However, path constraints cannot be directly imposed, as they need to be integrated from curvature. Based on [31], [32] presents a planner that first samples endpoints along the road and then connects them using curvature polynomials to make all paths conform to the road shape. After that, a set of trajectories is generated by specifying different acceleration profiles for each path. Paths generated in [32] can be tracked very well by an autonomous vehicle. [28] generates the lateral and longitudinal trajectory using quintic polynomials versus time, and it uses a Frenet frame referenced to the road center line to combine lateral and longitudinal motion. This makes the trajectory longitudinally coincide with the road shape. However, curvature has a complicated expression in the Frenet frame, making it inefficient to impose curvature constraints. A spline optimization method [33] that is based on Frenet frame has the same limitation for curvature. We solve this problem by defining the spline in global coordinates, where both the path and curvature constraints can be reformulated to be linear functions of the states, as discussed in Sec 3.5 and 3.6.

For some special cases [34, 35], analytic solutions exist for inverse kinematics. The Dubins car model [34] assumes the robot moves at constant forward speed, and it can either go straight or turn at the maximum steering angle. The Reeds-Shepp car model [35] is the same as the Dubins car except that it allows movement in the reverse direction. For Dubins and Reeds-Shepp cars, the shortest path between two states can always be expressed as combinations of motion primitives. Therefore, the solution can be computed efficiently. One limitation of both methods is that a fixed maximum steering angle needs to selected, which leads to a fixed turning radius. Another limitation is that the motion primitives are only straight lines or arcs, so the curvature is not continuous at the joint point of different motion primitives.

Both forward kinematics and inverse kinematics-based trajectory generation methods can be combined with a search algorithm for motion planning problems. The difference is, obviously, that the inverse method is designed to connect a start and final state [36] but the forward method doesn't guarantee reaching a particular final state [37]. In [36], differential constrained control sets are generated offline using the inverse method proposed in [31]. Such control sets not only satisfy differential constraints but also reach endpoints exactly. Therefore, resolution-completeness is attained when searching with the differential constrained control sets. The D* Lite [22] heuristic search algorithm is used to solve the planning problem.

2.3 Optimization-based Path Planning

Although optimization-based planning methods may converge to a local optimum when the problem is not convex, they have the advantages that the output trajectory can be very smooth due to its operating in continuous state space and they can converge quickly along the gradient direction, avoiding the curse of dimensionality.

2.3.1 Optimal Control and Trajectory Optimization

Trajectory optimization is a technique to provide an open-loop solution to an optimal control problem (OCP). A special case of the general nonlinear optimal control problem is the linear quadratic regulator (LQR), where the system dynamics can be described as linear differential equations and the cost can be described by a quadratic function. For a linear time-invariant (LTI) system, an analytic solution for LQR exists by directly solving the Riccati equation [38]. For a linear time-varying (LTV) system, the optimal policy can be computed recursively based on the Riccati equation. For nonlinear systems, iterative linear quadratic regulator (iLQR) [39, 40] uses an iterative version of LQR to find the optimal trajectory, where the first-order derivatives of the dynamics are used. The Differential Dynamic Programming (DDP) algorithm [41] is very similar to iLQR, except that the second-order derivatives of the system model are used during iterations. [42] proposes LQR-smoothing, which applies both a backward and forward LQR pass. It also optimizes the duration of a trajectory. In the original LQR setting, no constraints can be added for state and control variables. Both iLQR and DDP optimize only over the unconstrained control and state space. In [43], box inequality constraints for control variables are introduced in DDP. To handle the constraints on control, a lightweight quadratic program (QP) is solved at each backward pass. [44] combines an active-set method and Riccati-based method to handle linear inequality path constraints. To incorporate constraints such as static and dynamic obstacles into iLQR or DDP, an alternative way is to convert the constraints into cost with a barrier function [42, 45].

For the more general nonlinear optimal control problem with state and control constraints, direct trajectory optimization methods such as *shooting* [46] and

collocation [47] are commonly used. Direct methods first transform the trajectory optimization problem into a constrained nonlinear programming (NLP) problem. The NLP problem can then be solved by numerical optimization methods such as sequential quadratic programming (SQP) [48] or the interior point method [49, 50, 51]. The process of transforming an optimal control problem (OCP) to a nonlinear programming problem (NLP) is called *transcription*. Typical transcription approaches include *direct single shooting*, *direct multiple shooting*, and *direct collocation* [52]. In direct single shooting, we only discretize the controls, which are the decision variable in the nonlinear program, along the trajectory. The state trajectories are obtained by forward simulation of the system model [53]. Therefore, the system model constraints are automatically satisfied. Since the state trajectory is represented as an implicit function of the decision control variables, constraints on the state cannot be easily added. Solutions from the direct single shooting method are more sensitive to initialization than multiple shooting and collocation. An extension of direct single shooting is direct multiple shooting. In direct multiple shooting [48], the trajectory is divided into multiple segments. Each of the segments is represented in the same way as for direct single shooting. Equality constraints are imposed at the joint points of the segments to ensure continuity. For direct multiple shooting, additional state constraints can be added at the joint points as needed. Different from the shooting methods, in collocation, both the controls and the states are represented as piecewise polynomials. To satisfy the system dynamics, equality constraints on the derivative of the state polynomials are imposed [52]. For direct collocation, additional state and control constraints can be added at any discretized point as needed. Another advantage of the direct collocation method is that integrals and derivatives of state and control can be easily computed because they are in polynomial form.

Our quadratic spline formulation (3.13) can be viewed as a collocation method, and recurrent spline formulation (3.25) can be viewed as a multiple shooting method.

2.3.2 Model Predictive Control (MPC) based Motion Planning

Model predictive control (MPC) [54] is a strategy to generate feedback control based on an open-loop optimal control formulation. At each cycle, MPC repeatedly solves an optimal control problem (OCP) using a trajectory optimization approach. MPC has been widely applied to motion planning problems in recent years [55, 56, 57, 58, 59, 60].

[55] formulates the highway driving planning as a convex quadratic program (QP) optimization problem. It assumes the road is straight and the vehicle has a simplified linear motion model. For lane change trajectory planning, the collision avoidance constraints are designed as a linear formulation. CVXGEN [50] is used to generate and solve the QP.

In [56], a dynamic bicycle model with a simplified Pacejka Tire Model [61] is used. With the lateral tire friction model, the side slip effects can be calculated when the tire forces are saturated or close to saturation at handling limits. The racing problem with static obstacle avoidance is formulated as a nonlinear model predictive control (NMPC) problem. A high-level dynamic programming (DP)-based planner first generates a corridor, converting obstacles into a set of convex constraints. The NMPC problem is solved by locally linearizing the nonlinear bicycle model around the previous trajectory to obtain a linear time-varying (LTV) model. The resulting QP at each cycle is solved by an efficient interior point solver in FORCES [51, 62].

[57] proposes the convex feasible set algorithm to handle non-convex constraints. It iteratively finds a convex feasible set for the original non-convex prob-

lem. [58] uses a piecewise clothoid to represent vehicle motion, which allows sparse waypoints and leads to a small search space for the optimization problem. [59] considers the vehicle dimension constraints in the MPC framework. [60] uses a simple vehicle model derived from a nonlinear dynamic bicycle model that accounts for tire slip effects. It uses ACADO [63] to model and solve the MPC problem.

2.4 Speed Planning

Speed planning generates a speed profile for a given path, where a time-optimal speed profile is the most common goal. Speed planning plays a very important role in autonomous driving. In many scenarios, only speeding up and slowing down are needed when interacting with other participants on the road. Examples are following a leading car, stopping at a traffic light, yielding to cross traffic or pedestrians, and merging onto the highway. Speed planning on a fixed path is computationally more efficient than spatiotemporal planning. Speed planning can be obtained by a search-based method [64, 65, 66, 67] or optimization-based method [68, 69, 70, 71].

Two types of constraints need to be considered in speed planning. The first one is point-wise speed limit, for example, road speed limit and stop signs. It can be represented in the $s - v$ space, where s is the distance travel along the path, and v is the speed. The second type of constraint is imposed by the dynamic obstacles. It can be represented by forbidden regions of the st space. To simultaneously handle these two types of constraints, [65] directly searches in the state-time space, i.e., the $s - v - t$ space, which is not practical for realtime applications. [33] searches in the $s - t$ space using dynamic programming. To allow high speed in each expansion, the time resolution needs to be very small, making the search space large. [64] searches in the $s - v$ space using a visibility graph. The limitation is that it cannot

handle acceleration constraints. [66] extends [64] to handle acceleration constraints by constructing a dynamically reachable set in the $s-v-t$ space. In the absence of moving obstacles, [67] proposes a linear-time two-pass algorithm in the $s-v$ space, which generates the minimal-time solution given acceleration limits. By change of variables, [68, 69] formulate the speed optimization over a fixed path in a convex form. [70] extends [68] to handle some non-convex constraints. [71] takes into account dynamic obstacles in the optimization.

2.5 Summary

Table 2.1: Compared with different path optimization methods

Method	Kinematic constraints	Obstacles	Physical limit	Convex	Efficiency
Clothoid [31, 32]	✓	✗	✓	✗	++
Frenet frame [28]	✗	✓	✗	✓	+++
iLQR [42, 43]	✓	✗	✓	✗	++
MPC [56]	✓	✓	✓	✗	+
Proposed	✓	✓	✓	✓	+++

Table 2.2: Compared with different speed optimization methods

Method	Minimal time	Smoothness	Path constraint	Time window	Efficiency
Lipp *et al.* [69]	✓	✓	✓	✗	+++
Gu *et al.* [67]	✗	✗	✓	✗	+++
Fan *et al.* [33]	✗	✓	✗	✓	+++
Liu *et al.* [71]	✓	✓	✓	✓	+
Proposed	✓	✓	✓	✓	+++

For path optimization, handling the kinematic constraints and obstacles under the optimization framework is challenging. We develop a novel convex path optimization based on differential flatness theory, which automatically satisfies kine-

matic constraints. We compare our path optimizer with other methods in terms of handling kinematic constraints, dealing with obstacles, physical limits, convexity, and computational efficiency in Table 2.1. For speed optimization, most of the existing methods cannot simultaneously handle constraints on path and time dimensions except [71]. The drawback of [71] is that it is computationally expensive due to the nonlinearity of its formulation. We propose a novel speed path optimization through iterative quadratic programming to address this problem. We compare it to other methods in terms of the capability to handle time minimization, smoothness, path constraints, and time window constraints (when interacting with dynamic obstacles) in the optimization framework in Table 2.2.

Chapter 3

Path Optimization via Recurrent Spline Optimization

3.1 Introduction

In this chapter, we present a novel convex optimization-based approach for the autonomous vehicle motion planning problem. There are three types of constraints making the problem non-convex: obstacles, the nonlinear system model, and system limits (e.g. curvature constraint). First, based on differential flatness theory, we show that the nonlinear kinematic bicycle model can be automatically satisfied with a spline representation, which makes the nonlinear system constraints linear. Second, we build corridors based on vertical cell decomposition, which converts obstacles into convex constraints. Third, we handle the curvature constraint through iteratively convex optimization.

Our contributions are summarized as follows: 1) We convert motion planning into a convex quadratic program, which enables a reliably global optimal trajectory.

2) By reformulating the constrained spline optimization into optimal control form, we further reduce the computational time to be linear in the optimization horizon size. 3) Our method doesn't require an initial feasible solution, allowing an infeasible start. 4) Our approach allows large step size thanks to the linear system model based on differential flatness theory.

The rest of the chapter is organized as follows: The kinematic bicycle model and differential flatness are described in Sections 3.2 and 3.3 respectively. Constrained spline optimization is introduced in Section 3.5. Heading and curvature constraints are discussed in Section 3.6. Section 3.7 presents the reformulation of the constrained spline optimization into optimal control. Experiments for trajectory generation and planning are described in Section 3.8.

3.2 Kinematic Bicycle Model

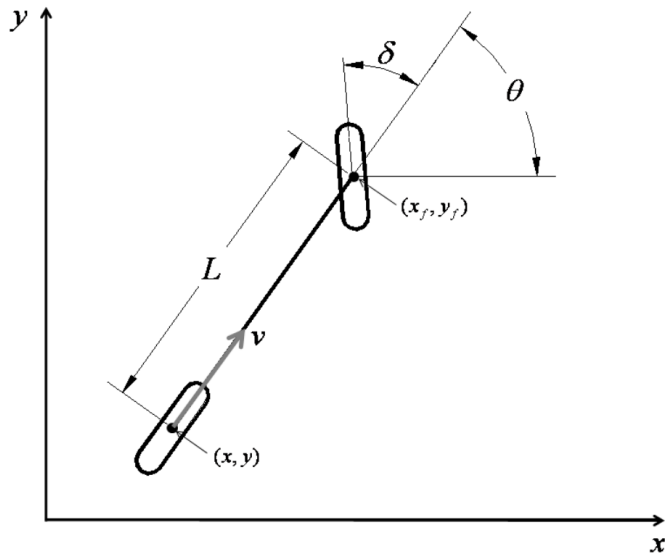


Figure 3.1: Kinematic bicycle model

The kinematic model is subject to nonholonomic constraints due to the rolling without slipping condition between the wheels and the ground [72]. The front wheel is steerable and the rear wheel orientation is fixed. As shown in Fig. 3.1, (x, y) are the global coordinates of the rear wheel, (x_f, y_f) are the global coordinates of the front wheel, θ is the orientation of the car body in the global frame, δ is the steering angle in the body frame, ω is the angular velocity of the steering wheel, L is the distance between the wheels, and v is the longitudinal velocity along the body at the rear wheel. The state of the kinematic model is defined as $\mathbf{x} = (x, y, \theta, \delta)$, and the control input is $\mathbf{u} = (v, \omega)$. With the no-lateral-slip assumption, the lateral velocity at the rear wheel is zero, which gives

$$\begin{aligned} \dot{x} &= v\cos(\theta) \\ \dot{y} &= v\sin(\theta) \end{aligned} \tag{3.1}$$

The nonholonomic constraint for the front wheel is

$$\dot{x}_f \sin(\theta + \delta) - \dot{y}_f \cos(\theta + \delta) = 0 \tag{3.2}$$

By using the rigid-body constraint

$$\begin{aligned} x_f &= x + L\cos(\theta) \\ y_f &= y + L\sin(\theta) \end{aligned} \tag{3.3}$$

The kinematic constraint for the front wheel (3.2) becomes

$$\dot{x}\sin(\theta + \delta) - \dot{y}\cos(\theta + \delta) - \dot{\theta}L\cos(\delta) = 0 \tag{3.4}$$

Substituting (3.1) into (3.4) yields

$$\dot{\theta} = v \tan \delta / L \tag{3.5}$$

The system equation of a kinematic car model $\dot{\mathbf{x}} = f_k(\mathbf{x}, \mathbf{u})$ is

$$\begin{aligned} \dot{x} &= v \cos \theta \\ \dot{y} &= v \sin \theta \\ \dot{\theta} &= v \tan \delta / L \\ \dot{\delta} &= \omega \end{aligned} \tag{3.6}$$

3.3 Differential Flatness

3.3.1 Differential Flatness Theory

A system is differentially flat if there exists an output $y \in \mathbb{R}^m$ (as a function of the state, input, and input derivatives), called flat output, such that the state $x \in \mathbb{R}^n$ and the control input $u \in \mathbb{R}^m$ of the system can be expressed as algebraic functions of the flat output y and its time derivatives up to a certain order [73, 74]. Roughly speaking, a system is flat if there exists a set of outputs (equal in number to the number of control inputs) such that all states and control inputs can be determined from these outputs without integration. Mathematically, the system is flat if we can find output y of the form

$$y = \gamma(x, u, \dot{u}, \ldots, u^{(r)})$$

such that

$$\begin{aligned} x &= \alpha(y, \dot{y}, \ldots, y^{(q)}) \\ u &= \varphi(y, \dot{y}, \ldots, y^{(q)}) \end{aligned}$$

As a consequence, once a flat output trajectory y is assigned, the associated trajectory of the state x and control inputs u are uniquely determined. The resulting trajectory of x will automatically satisfy the nonholonomic constraints.

3.3.2 Differential Flatness for the Kinematic Bicycle Model

For the kinematic bicycle model $\dot{\mathbf{x}} = f_k(\mathbf{x}, \mathbf{u})$ defined in (3.6), the state is $\mathbf{x} = (x, y, \theta, \delta)$, and the control input is $\mathbf{u} = (v, \omega)$. We will show that the global coordinates of the rear wheel (x, y), which have the same dimension of the control input $\mathbf{u}$, are flat outputs of the kinematic bicycle system.

Assuming the car only moves forward, solving (3.1) gives the longitudinal velocity

$$v = \sqrt{\dot{x}^2 + \dot{y}^2} \tag{3.7}$$

From (3.1), we can also obtain the orientation of the car

$$\theta = \operatorname{atan2}(\dot{y}, \dot{x}) \tag{3.8}$$

Differentiating (3.8) yields

$$\dot{\theta} = \frac{\ddot{y}\dot{x} - \ddot{x}\dot{y}}{\dot{x}^2 + \dot{y}^2} \tag{3.9}$$

Plugging (3.9) into (3.5) gives

$$\delta = \arctan \frac{(\ddot{y}\dot{x} - \ddot{x}\dot{y})\, L}{(\dot{x}^2 + \dot{y}^2)^{\frac{3}{2}}} \tag{3.10}$$

Differentiating (3.10) yields

$$\omega = \frac{(\dddot{y}\dot{x} - \dddot{x}\dot{y})(\dot{x}^2 + \dot{y}^2) - 3(\ddot{y}\dot{x} - \ddot{x}\dot{y})(\dot{x}\ddot{x} + \dot{y}\ddot{y})}{(\dot{x}^2 + \dot{y}^2)^3 + L^2(\ddot{y}\dot{x} - \dddot{x}\dot{y})^2}\sqrt{\dot{x}^2 + \dot{y}^2}L \tag{3.11}$$

So far we have shown that, given output trajectory (x, y) and their derivatives up to the third order, we can reproduce other states (θ, δ) in (3.8) (3.10) and control inputs (v, ω) in (3.7) (3.11). Therefore, the kinematic bicycle model is differentially flat. As a consequence, if the trajectory (x, y) is three times differentiable, it will automatically satisfy the nonholonomic kinematic constraints.

3.4 Handling Obstacles with Corridor

For on-road driving scenarios, the number of traffic participants that may interact with the autonomous vehicle is often limited. Therefore, instead of applying dynamic programming to a lattice grid [56], we use vertical cell decomposition (VCD) to partition the free-space into a set of cells. Each cell is either a trapezoid or a triangle. The corridor is built by a list of connected convex cells. The algorithm has computational efficiency of $O(n \log n)$, where n is the number of vertices in the environment.

We assume static obstacles are given by the perception system. The uncertainties from the perception system are not handled in this work. We assume the changing of the environment is handled by the re-planning scheme. The noise of the state of obstacles within one planning cycle can be handled by inflating the obstacles.

3.5 Constrained Spline Optimization

We have shown in Section 3.3.2 that the Kinematic Bicycle Model is differentially flat. Therefore, we can use a polynomial to represent the path. Based on (3.11), the polynomial needs to be at least three times differentiable. We choose a parametric form $(x, y) = (f(s), g(s))$ to represent a curve of any shape, where s is the arc length of the reference path, and $f(s)$ and $g(s)$ are cubic splines defined by piecewise-cubic polynomials. It is notable that s is not the arc length of the final trajectory because the latter is unknown until the optimization converges.

As mentioned in the last paragraph, we assume a reference path is given. The reference path can be chosen as the center line of the lane or the straight line connecting start and goal state depending on different applications. It is notable that, unlike many other methods, the reference path doesn't need to be collision-free with obstacles in our approach. The points from the reference path are denoted as $\{(s_i, \bar{x}_i, \bar{y}_i, \bar{\theta}_i),\ i = 0, 1, \ldots, N\}$. The i^{th} piece of the spline is defined below, where $s_i \leq s < s_{i+1}$.

$$\begin{aligned} f_i(s) &= a_{i0} + a_{i1}(s - s_i) + a_{i2}(s - s_i)^2 + a_{i3}(s - s_i)^3 \\ g_i(s) &= b_{i0} + b_{i1}(s - s_i) + b_{i2}(s - s_i)^2 + b_{i3}(s - s_i)^3 \end{aligned} \tag{3.12}$$

We formulate the planning problem as a constrained spline optimization problem. Different from the smoothing splines proposed in [75], where the constraints can only be added to the total differences between the smoothed path and the original path, our formulation allows imposing the constraints on any point. The resulting optimization is a constrained quadratic program.

The optimization problem is formulated below

$$\min_{a_{ij}, b_{ij}} \quad \sum_{i=0}^{N} w_1(\ddot{f}_i(s_i)^2 + \ddot{g}_i(s_i)^2) + w_2(\dddot{f}_i(s_i)^2 + \dddot{g}_i(s_i)^2) \tag{3.13a}$$

s.t.

$$f_0(s_0) = x_{init}, \quad g_0(s_0) = y_{init}, \tag{3.13b}$$

$$f_N(s_N) = x_{goal}, \quad g_N(s_N) = y_{goal}, \tag{3.13c}$$

$$f_i(s_i) = f_{i+1}(s_i), \quad g_i(s_i) = g_{i+1}(s_i), \tag{3.13d}$$

$$\dot{f}_i(s_i) = \dot{f}_{i+1}(s_i), \quad \dot{g}_i(s_i) = \dot{g}_{i+1}(s_i), \tag{3.13e}$$

$$\ddot{f}_i(s_i) = \ddot{f}_{i+1}(s_i), \quad \ddot{g}_i(s_i) = \ddot{g}_{i+1}(s_i), \tag{3.13f}$$

$$-\Delta_l \leq -(f_i(s_i) - \bar{x}_i)\sin\bar{\theta}_i + (g_i(s_i) - \bar{y}_i)\cos\bar{\theta}_i \leq \Delta_r, \tag{3.13g}$$

$$-\Delta_b \leq (f_i(s_i) - \bar{x}_i)\cos\bar{\theta}_i + (g_i(s_i) - \bar{y}_i)\sin\bar{\theta}_i \leq \Delta_f \tag{3.13h}$$

The optimization objective 3.13a is the square sum of the second-order derivatives of position (x, y), which stands for maximizing smoothness and minimizing energy. w_1 and w_2 are weighting parameters. Constraints (3.13b) and (3.13c) are the boundary conditions for start and goal positions. Sometimes, the goal is defined as a region instead of a fixed position, and (3.13c) can be modified to affine inequality constraints in this case. (3.13d)-(3.13f) ensure x and y and their first-order derivatives and second-order derivatives are continuous at the joints. Based on the differential flatness theory discussed in 3.3.2, these continuity constraints ensure the system kinematic constraints are satisfied. (3.13g) defines two affine inequality constraints at each point (one for the right and one for the left border of the corridor). (3.13h) defines another two affine inequality constraints at each point to limit the longitudinal movements. The resulting region is a convex feasible set.

The optimization problem defined in (3.13) contains a convex quadratic objective function and linear equality and inequality constraints. Thus, it is a constrained convex quadratic program, and the global optimal solution can be obtained by any QP solver (e.g. OSQP [76]) or the primal-dual interior point method in polynomial time [77, 78]. It has computational complexity of $O(N^3(n_p + n_c)^3)$, where N is the number of steps, n_p is the number of coefficients to be optimized at each step ($n_p = 8$ in problem (3.13)), and n_c is the number of constraints at each step.

3.6 Heading and Curvature Constraints

For path planning, it is critical to ensure that heading and curvature be continuous at the start point, and that curvature be within the maximum curvature range along the path. Given heading θ_0 at the starting point, the constraint for heading can be expressed as a linear function of $\dot{f}$ and $\dot{g}$ based on (3.8).

$$-\dot{f}_0(s_0)\sin\theta_{init} + \dot{g}_0(s_0)\cos\theta_{init} = 0 \tag{3.14}$$

For curvature $\kappa = \frac{\dot{y}\dot{x}-\ddot{x}\dot{y}}{(\dot{x}^2+\dot{y}^2)^{\frac{3}{2}}}$, denoting $\lambda = -\frac{\dot{y}}{(\dot{x}^2+\dot{y}^2)^{\frac{3}{2}}}$ and $\mu = \frac{\dot{x}}{(\dot{x}^2+\dot{y}^2)^{\frac{3}{2}}}$, we get

$$\kappa = \lambda\ddot{x} + \mu\ddot{y} \tag{3.15}$$

We observe that curvature κ is a linear function of the second-order derivatives $\ddot{x}$ and $\ddot{y}$ given the first-order derivatives $\dot{x}$ and $\dot{y}$. Therefore, we can design an iterative scheme to approximate curvature. In each iteration, λ and μ are determined by the value of $\dot{x}$ and $\dot{y}$ that are obtained from the last iteration. Curvature converges to the true value as the path converges to the final optimized path.

The only special case is that $\dot{x}$ and $\dot{y}$ are not available in the first iteration.

In this case, we will show how to approximate curvature with the reference path. Denote the arc length of the final optimized path as σ. By the Pythagorean theorem, we get $d\sigma = \sqrt{dx^2 + dy^2}$. Taking derivatives on both sides with respect to s, we get

$$\frac{d\sigma}{ds} = \sqrt{\left(\frac{dx}{ds}\right)^2 + \left(\frac{dy}{ds}\right)^2} = \sqrt{\dot{x}^2 + \dot{y}^2} \tag{3.16}$$

If σ is close to the arc length of the reference path s, we obtain $\frac{d\sigma}{ds} \approx 1$. Therefore,

$$\sqrt{\dot{x}^2 + \dot{y}^2} \approx 1 \tag{3.17}$$

If the heading from the reference path $\bar{\theta}$ is also close to the heading of the final path θ, we get $\bar{\theta} \approx \theta$. Substituting it and (3.17) into λ, we get

$$\lambda = \frac{-\dot{y}}{(\dot{x}^2 + \dot{y}^2)^{\frac{3}{2}}} = \frac{-1}{\dot{x}^2 + \dot{y}^2} \frac{\dot{y}}{\sqrt{\dot{x}^2 + \dot{y}^2}} \approx -\sin\bar{\theta} \tag{3.18}$$

Similarly, we get $\mu \approx \cos\bar{\theta}$. Substituting (3.18) into (3.15), yields that curvature can be approximated by the heading of the reference path in the first iteration of the optimization.

$$\kappa \approx -\ddot{x}\sin\bar{\theta} + \ddot{y}\cos\bar{\theta} \tag{3.19}$$

With (3.15) and (3.19), the curvature at the start point can be expressed as a linear equality constraint, and the curvature limit along the path can be expressed as linear inequality constraints. We have shown in (3.14) that the heading constraint is also a linear equality constraint. Therefore, with the additional heading and curvature constraints introduced in this section, the optimization problem (3.13) is still a constrained convex quadratic program.

3.7 From Constrained Spline Optimization to Optimal Control

In this section, we reformulate the constrained spline optimization in (3.13) to an optimal control problem. The special structure in the optimal control problem can then be exploited, yielding a more efficient solver.

3.7.1 Optimal Control Formulation

Denoting $h = s_{i+1} - s_i$, based on (3.12) and its derivatives, we get

$$\begin{aligned} x_{i+1} &= f_i(s_{i+1}) = a_{i0} + a_{i1}h + a_{i2}h^2 + a_{i3}h^3 \\ \dot{x}_{i+1} &= \dot{f}_i(s_{i+1}) = a_{i1} + 2a_{i2}h + 3a_{i3}h^2 \\ \ddot{x}_{i+1} &= \ddot{f}_i(s_{i+1}) = 2a_{i2} + 6a_{i3}h \\ \dddot{x}_{i+1} &= \dddot{f}_i(s_{i+1}) = 6a_{i3} \end{aligned} \tag{3.20}$$

Reorganize (3.20) into matrix form yields

$$\begin{bmatrix} x_{i+1} \\ \dot{x}_{i+1} \\ \ddot{x}_{i+1} \\ \dddot{x}_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & h & h^2 & h^3 \\ 0 & 1 & 2h & 3h^2 \\ 0 & 0 & 2 & 6h \\ 0 & 0 & 0 & 6 \end{bmatrix} \begin{bmatrix} a_{i0} \\ a_{i1} \\ a_{i2} \\ a_{i3} \end{bmatrix} \tag{3.21}$$

Based on (3.12), we also have

$$\begin{aligned} x_i &= f_i(s_i) = a_{i0}, & \dot{x}_i &= \dot{f}_i(s_i) = a_{i1} \\ \ddot{x}_i &= \ddot{f}_i(s_i) = 2a_{i2}, & \dddot{x}_i &= \dddot{f}_i(s_i) = 6a_{i3} \end{aligned} \tag{3.22}$$

Substituting (3.22) into (3.21) and denoting $\tilde{x}_i = (x_i, \dot{x}_i, \ddot{x}_i)$, $\tilde{u}_i^x = \dddot{x}_i$ yield

$$\tilde{x}_{i+1} = A\tilde{x}_i + B\tilde{u}_i$$
$$A = \begin{bmatrix} 1 & h & \frac{1}{2}h^2 \\ 0 & 1 & h \\ 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} \frac{1}{6}h^3 \\ \frac{1}{2}h^2 \\ h \end{bmatrix} \tag{3.23}$$

The linear system in (3.23) can be readily extended to higher dimensions by denoting $\tilde{y}_i = (y_i, \dot{y}_i, \ddot{y}_i)$, $\tilde{u}_i^y = \dddot{y}_i$, $\tilde{\mathbf{x}}_\mathbf{i} = (\tilde{x}_i, \tilde{y}_i)$, $\tilde{\mathbf{u}}_\mathbf{i} = (\tilde{u}_i^x, \tilde{u}_i^y)$.

$$\tilde{\mathbf{x}}_{\mathbf{i+1}} = \tilde{A}\tilde{\mathbf{x}}_\mathbf{i} + \tilde{B}\tilde{\mathbf{u}}_\mathbf{i}$$
$$\tilde{A} = \begin{bmatrix} A & & \\ & & A \end{bmatrix} \quad \tilde{B} = \begin{bmatrix} B & \\ & B \end{bmatrix} \tag{3.24}$$

By changing the optimization variables in (3.13) from spline coefficients (a_{ij}, b_{ij}) to $(\tilde{\mathbf{x}}_\mathbf{i}, \tilde{\mathbf{u}}_\mathbf{i})$, and replacing the continuity constraints defined in (3.13d)-(3.13f) to the LTI system (3.24), we can reformulate (3.13) as an optimal control problem as follows

$$\min_{\tilde{\mathbf{x}}_\mathbf{i},\, \tilde{\mathbf{u}}_\mathbf{i}} \quad \sum_{i=0}^{N} w_1 \left(\dddot{x}_i^2 + \dddot{y}_i^2\right) + w_2 \left(\ddddot{x}_i^2 + \ddddot{y}_i^2\right) \tag{3.25a}$$

$$\text{s.t.} \quad x_0 = x_{init}, \quad y_0 = y_{init}, \tag{3.25b}$$

$$x_N = x_{goal}, \quad y_N = y_{goal}, \tag{3.25c}$$

$$\tilde{\mathbf{x}}_{\mathbf{i+1}} = \tilde{A}\tilde{\mathbf{x}}_\mathbf{i} + \tilde{B}\tilde{\mathbf{u}}_\mathbf{i}, \tag{3.25d}$$

$$-\Delta_l \leq -(x_i - \bar{x}_i)\sin\bar{\theta}_i + (y_i - \bar{y}_i)\cos\bar{\theta}_i \leq \Delta_r, \tag{3.25e}$$

$$-\Delta_b \leq (x_i - \bar{x}_i)\cos\bar{\theta}_i + (y_i - \bar{y}_i)\sin\bar{\theta}_i \leq \Delta_f, \tag{3.25f}$$

$$-\dot{x}_0 \sin\theta_{init} + \dot{y}_0 \cos\theta_{init} = 0, \tag{3.25g}$$

$$\lambda_0 \ddot{x}_0 + \mu_0 \ddot{y}_0 = \kappa_{init}, \tag{3.25h}$$

$$\underline{\kappa} \leq \lambda_i \ddot{x}_i + \mu_i \ddot{y}_i \leq \bar{\kappa} \tag{3.25i}$$

As in (3.13), (3.25b) and (3.25c) are boundary constraints, (3.25d) is the linear system constraint, and (3.25e) and (3.25f) define two affine constraints for lateral and longitudinal offset, respectively. Additionally, (3.25g) and (3.25h) respectively define constraints for heading and curvature at the starting state. (3.25i) defines the curvature constraints along the path. As denoted in (3.15), (λ_i, μ_i) are functions of $\dot{x}$ and $\dot{y}$, and their values are updated through iterations except that they are approximated by the reference path at the first iteration.

3.7.2 Computational Complexity for Optimal Control Problem

Next, we will show that the convex quadratic program (3.25) can be solved in $O(N(n_x+n_u+n_c)^3)$, where N is the horizon length, n_x and n_u are number of states and control inputs, n_c is the number of constraints other than the system model at each step. A brief introduction to the primal-dual interior point method is given here. More details can be found in [49, 79]. The problem defined in (3.25) can be rearranged into a generic convex quadratic form.

$$\begin{aligned} \min_{w} \quad & \frac{1}{2} w^T Q w + c^T w \\ \text{s.t.} \quad & Fw = g, \\ & Cw \leq d \end{aligned} \tag{3.26}$$

The KKT conditions yield

$$\begin{aligned} Qw + c + F^T\pi + C^T\lambda &= 0 \\ -Fw + g &= 0 \\ -Cw + d - t &= 0 \\ T\Lambda e &= 0 \\ (\lambda, t) &\geq 0 \end{aligned} \tag{3.27}$$

where π and λ are the Lagrange multipliers of the equality and inequality constraints, respectively. t is a vector of slack variables for the inequality constraints. $T = \text{diag}(t_1, t_2, \ldots, t_n)$, $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$, $e = (1, 1, \ldots, 1)^T$, and n is the number of inequality constraints. The primal-dual interior point method uses Newton's method to solve the system $f(w, \pi, \lambda, t) = 0$ defined in (3.27) iteratively. The Newton direction is computed by solving the following linear system.

$$\begin{bmatrix} Q & F^T & C^T & 0 \\ -F & 0 & 0 & 0 \\ -C & 0 & 0 & -I \\ 0 & 0 & T & \Lambda \end{bmatrix} \begin{bmatrix} \Delta w \\ \Delta\pi \\ \Delta\lambda \\ \Delta t \end{bmatrix} = \begin{bmatrix} r_Q \\ r_F \\ r_C \\ r_T \end{bmatrix} \tag{3.28}$$

The right-hand-side vector is denoted as (r_Q, r_F, r_C, r_T), which decides different search directions. The basic primal-dual method uses pure Newton direction. The Mehrotra predictor-corrector algorithm combines Newton and centering directions. By eliminating $\Delta\lambda$ and Δt in (3.28), a reduced KKT system is obtained.

$$\begin{bmatrix} Q + C^T\Lambda T^{-1}C & F^T \\ -F & 0 \end{bmatrix} \begin{bmatrix} \Delta w \\ \Delta\pi \end{bmatrix} = \begin{bmatrix} r_Q - C^T T^{-1}(\Lambda r_C + r_T) \\ r_F \end{bmatrix} \tag{3.29}$$

For the linear-quadratic optimal control problem (3.25), (3.29) can be rewritten in block tridiagonal form [49]. The block tridiagonal matrix allows the linear system (3.29) to be solved in $O(N(n_x + n_u + n_c)^3)$ using the Schur complement-based method (e.g. fast-mpc [80], FORCES [51]) or Riccati-based method (e.g. hpipm [81]), as opposed to $O(N^3(n_x + n_u + n_c)^3)$ for KKT systems with generic matrix form.

3.8 Experimental Results

The proposed algorithms are tested for two types of problems: trajectory generation and path planning. Trajectory generation is a boundary value problem where only the start and goal state, and the system model and limit are considered. Additionally, path planning also handles obstacles. For each problem, two scenarios are included: lane change and sharp turn. We compare the two methods proposed in this chapter (constrained spline optimization and linear-quadratic optimal control) to four other methods: κ-poly (curvature polynomial from [31]), iLQR (extended LQR from [42]), acado (ACADO [63] with default settings, an active-set based multiple-shooting approach), and ms-ipopt [82] (an interior-point based multiple-shooting approach). The kinematic bicycle model is used for these four methods. The tests are implemented on a Desktop PC with Intel i7-9700K CPU (8 core 3.60GHz) and 32GB memory.

3.8.1 Trajectory Generation

We first compare the runtime for a boundary problem with only start and goal state as constraints in Table 3.1, where N is the number of steps. The states consist of 2D position, heading, and curvature. The number of optimization variables is fixed

Table 3.1: Runtime comparison for trajectory generation with different horizons.

	Lane Change [ms]			Sharp Turn [ms]		
N	10	40	160	10	40	160
κ-poly [31]	0.32	0.32	0.38	0.41	0.42	0.46
iLQR [42]	2.18	3.13	4.63	failed	2.41	8.31
ms-ipopt [82]	3.24	4.82	11.26	7.62	17.64	11.66
acado [63]	9.29	56.22	923.41	16.59	72.16	1275.74
spline-qp	0.46	1.94	9.29	0.42	1.53	8.23
spline-ocp	0.08	0.21	0.62	0.11	0.24	0.69

Table 3.2: Runtime comparison for trajectory generation with vs. without curvature constraints for N=40.

	Lane Change [ms]		Sharp Turn [ms]	
Curvature	w/o	w/	w/o	w/
κ-poly [31]	0.32	N/A	0.42	N/A
iLQR [42]	3.13	N/A	2.41	N/A
ms-ipopt [82]	4.82	6.43	4.24	8.14
acado [63]	56.22	68.63	72.16	102.33
spline-qp	1.94	6.24	1.53	18.83
spline-ocp	0.21	0.57	0.24	1.14

for the curvature polynomial method (denoted as κ-poly), and is linear in N for all other methods.

Runtime for κ-poly grows slowly with N, as only integration steps are increased. The control input (curvature) in κ-poly is represented as a single polynomial, which reduces the number of variables to be optimized. However, the trade-off is that it also reduces the flexibility of the trajectory. It may be unable to generate trajectories with desirable curvature at some sharp turns. In contrast, our spline-based method can generate curves of any shape.

The acado method is the slowest one among all methods because the condensing technology it uses is not beneficial for our problem type. The active-set method used by acado also does not scale well with horizon. The acado method is more suitable for control problems where the needed horizon is much less than that for the planning problem.

The iLQR method fails to converge for the sharp turn test with $N = 10$. This is because as N decreases, the gap between two consecutive states increases. Therefore, iLQR becomes more sensitive to the integration error due to linearizing the nonlinear kinematic model. In contrast, our spline-based methods have a linear system model thanks to differential flatness theory, which allows large gaps between states.

The ms-ipopt method takes longer for $N = 40$ vs. 160 in the sharp turn test. This is another example of the NLP problem in iLQR and ms-ipopt having a less consistent convergence rate vs. our convex QP formulation. The proposed spline-ocp method has lowest runtime at $N = 10$, and it grows linearly with horizon N.

Table 3.2 shows the runtime when we add curvature constraints along the

path. The values for iLQR and κ-poly are not available, since they cannot add curvature constraints. For the proposals method, it usually takes 2-3 iterations for difficult cases to satisfy the curvature constraints as it reflects in the runtime.

3.8.2 Path Planning

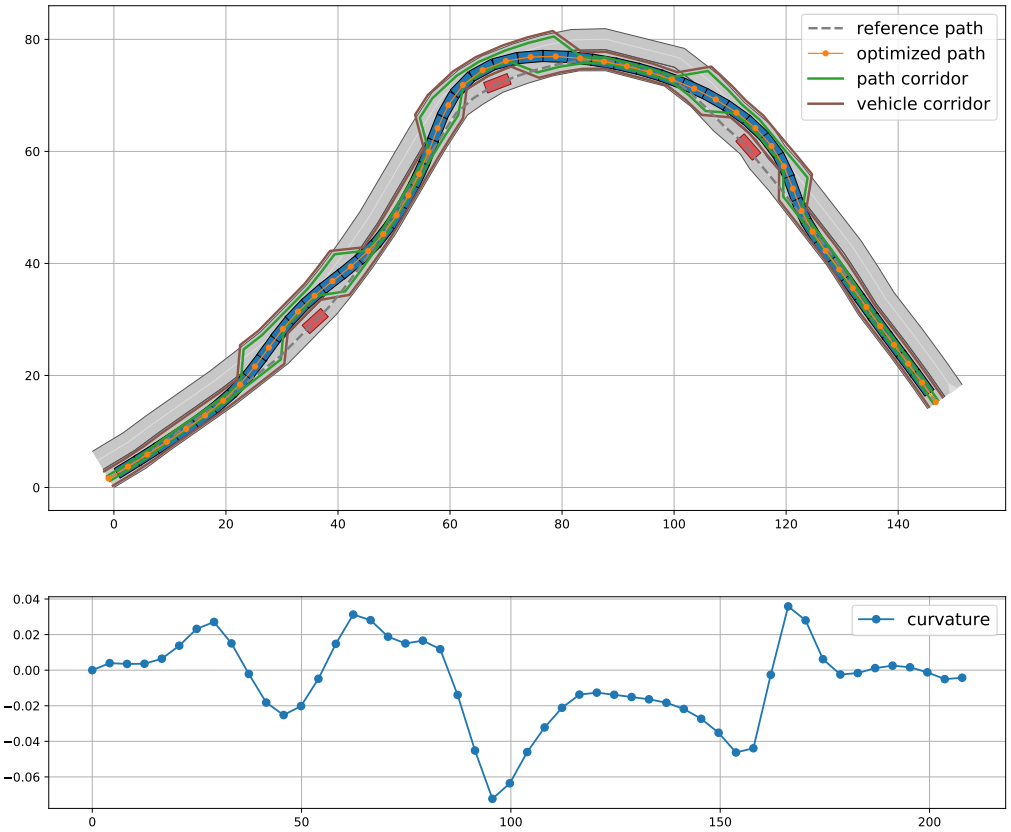


Figure 3.2: Avoiding static obstacles on a curvy road.

We also compare runtime for path planning with obstacles. κ-poly and iLQR are not included again, since their formation do not allow path constraints. A collision-free corridor is first generated based on vertical cell decomposition, as described in 3.4. The constraints formed by the corridor are convex for each state. The

results are shown in Table 3.3. Both acado and ms-ipopt fail to converge at $N = 10$ in the lane change test. This is due to a combination of the large linearization error as explained before, as well as the curvature and obstacle constraints making the problem harder to solve.

Fig 3.2 shows a scenario where the autonomous vehicle avoids multiple obstacles on a curvy road. In this example, it takes 0.54 ms to finish the optimization for a 200 m path with $N = 50$. The average gap between two states is 4 m, which is much larger than a normal MPC setting. This is thanks to the linear spline model in our method, as mentioned before. Fig 3.3 shows the autonomous vehicle making a sharp turn while staying in lane with curvature within limit. The planned path cuts the corner of the corridor nicely to reach the minimal turning radius. The two scenarios in Fig 3.2 and Fig 3.3 are from the CommonRoad benchmark [83].

Table 3.3: Runtime comparison for path planning with obstacles.

	Lane Change [ms]			Sharp Turn [ms]		
N	10	40	160	10	40	160
ms-ipopt [82]	failed	6.78	19.96	5.09	9.74	34.48
acado [63]	failed	68.43	1123.14	18.44	142.85	9428.77
spline-qp	2.15	9.76	46.85	2.92	22.42	397.76
spline-ocp	0.27	0.63	2.92	0.32	1.35	6.76

3.9 Conclusions

Our work is inspired by exploring the relationship between smoothing spline [75], iLQR [39], and multiple shooting-based trajectory optimization [46]. It can be viewed as an extension of the smoothing spline (with additional path and curvature constraints), with a multiple shooting formulation, and solved in a LQR way. It

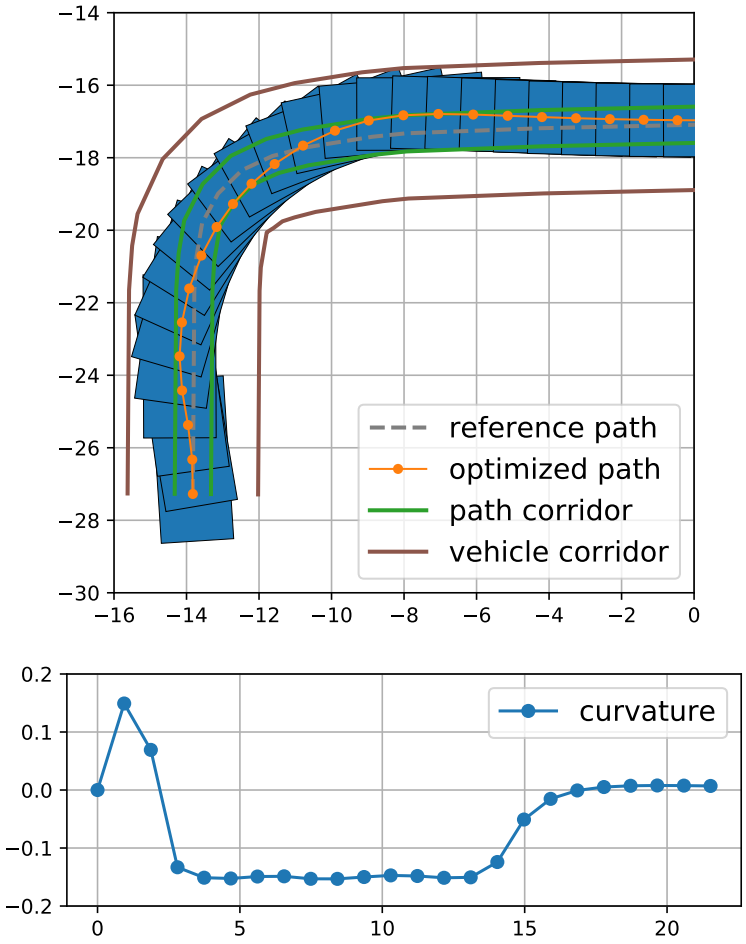


Figure 3.3: Making a sharp turn while staying in lane.

provides a new way to solve a normal spline smoothing problem. It can also be used to smooth the reference path in addition to trajectory generation and motion planning applications shown in this chapter.

The way we use the reference path is different from that of many other methods. Unlike most of the other planning approaches, the reference path in our method also doesn't need to be collision-free. We use the reference to help with defining obstacle and curvature constraints, and it can be easily obtained. For example, the reference path can be chosen as the straight line connecting start and goal state for trajectory generation problems, and the center line of the road for motion planning problems.

Several directions may be explored as future work. Our method can potentially be extended to trajectory planning with dynamic obstacles by replacing the arc length s with time t. However, it is hard to guess the final time horizon before the optimization is finished. Therefore, the total time also needs to be optimized, which makes the problem very nonlinear and non-convex. It is also difficult to convert the dynamic obstacles into convex constraints. Our method can also be extended to 3D space similar to the extension from 1D to 2D in (3.24).

Chapter 4

Speed Planning in Dynamic Environments over a Fixed Path

4.1 Introduction

To reduce the complexity of trajectory planning in dynamic environments, the task can often be decomposed into two sub-tasks. The first sub-task is to find a collision-free path among static obstacles. The second sub-task, which is the focus of this chapter, is to generate a time-optimal speed profile on the given path while keeping a safe distance from dynamic obstacles and satisfying vehicle dynamics (speed and acceleration constraints). In some typical autonomous driving scenarios, such as following a leading car, handling an unprotected left turn, and merging onto a main road, there is little room to improve performance by adjusting the path, whereas speed planning can play a critical role in interacting safely with dynamic obstacles.

The motion along a fixed path can be described by the time coordinate t, the single path coordinate s, and its time derivatives, the speed $\dot{s}$ and the acceleration

$\ddot{s}$. The dynamic obstacles can be described as forbidden regions in the path-time ($s \times t$) space by computing the intersections with the path, which is similar to the handling of static obstacles in the path planning problem. In the path-time space, the speed planning problem can be viewed as a special path planning problem with additional constraints. First, the time needs to be monotonically increasing. Second, the speed and acceleration constraints need to be satisfied. For autonomous driving, speed planning often comprises three goals and four constraints, in addition to the implicit monotonicity time constraints ($c0$). The three goals are minimizing time ($g1$), maximizing smoothness ($g2$), and keeping a safe distance from the leading car ($g3$). The four constraints are dynamic obstacles ($c1$), speed ($c2$), acceleration ($c3$), and the system dynamics ($c4$) constraints.

Methods for speed planning can be roughly divided into three categories: search-based methods ([64, 66]), optimization-based methods ([68, 84, 71]), and switch-point methods ([85, 86, 87]).

The search-based method in [64] uses a visibility graph to solve the minimum-time ($g1$) problem with dynamic obstacles ($c1$) and speed constraints ($c2$). [66] extend this method to handle acceleration constraints ($c3$). One limitation of both [64] and [66] is that they only accept single, global speed and acceleration constraints but not point-wise constraints. For autonomous driving, point-wise speed constraints are needed on curvy roads or when making turns to limit lateral centripetal force to prevent rollover. Another limitation of search-based methods is that they cannot directly maximize smoothness ($g2$). The speed profiles generated by search-based methods need to be further smoothed to ensure driving comfort. The smoothing procedure is not trivial in some challenging cases where the solution from the search-based method is close to the constraint limits, since the smoothing itself becomes a

constrained optimization problem which is hard to solve.

Optimization-based methods generally allow point-wise constraints for speed ($c1$) and acceleration ($c2$), and are also able to directly maximize smoothness ($g2$). However, dynamic obstacles are difficult to handle by optimization-based methods because they are non-convex constraints in the $s \times t$ space. [68] address the time-optimal path-following problem, where the goal is to find the time-optimal ($g1$) speed profile with speed ($c1$) and acceleration ($c2$) constraints. It reformulates the original problem as a convex optimization by a nonlinear change of variables, which guarantees a global optimum. However, their formulation becomes non-convex with dynamic obstacle constraints ($c3$). [84] extend this method to handle dynamic obstacles. However, the constraints for dynamic obstacles can only be imposed on one side in their formulation to keep the problem convex. More precisely, they can only handle the case when the autonomous vehicle needs to reach a position earlier than dynamic obstacles but not vice versa. [71] propose a new formulation that uses time as a decision variable, and the problem is then iteratively solved as a quadratic program. This method can handle all goals ($g1$ and $g2$) and point-wise constraints ($c1$–$c4$). The only drawback is that it is computationally expensive due to the nonlinearity of its formulation.

Switch-point methods can also be called bang-bang approaches. The basic idea comes from [85], which proves that for the time-optimal problem ($g1$) the optimal solution always takes either the maximum acceleration or deceleration, which is similar to bang-bang control. Therefore, the problem reduces to finding the switch-points to alternate between the maximum acceleration and deceleration ([86, 87]). The switch-point methods are theoretically faster than the other two types of methods. However, their exclusive use of either maximum acceleration or deceleration

does not hold if other goals are added ($g2$ and $g3$). Thus, they are not suitable for the problems considered in this chapter.

In this chapter, we propose a novel speed planning approach to handle all constraints discussed above efficiently. First, we present an iterative convex optimization formulation that can simultaneously impose dynamic obstacles and pointwise speed constraints. Second, we propose a modified vertical cell decomposition method that divides the $s \times t$ space into collision-free channels, and converts the dynamic obstacles into convex constraints. Third, we introduce an optimization-based method and a linear-time method to generate initial solutions for speed optimization.

The rest of the chapter is organized as follows. The path representation and discretization, objective function, and constraints for speed planning of autonomous driving are presented in Sections 4.2.1, 4.2.2, and 4.2.3, respectively. A time-discretization formulation and a space-discretization formulation for speed optimization from the literature are introduced in Sections 4.2.4 and 4.2.5, respectively. Their advantages and disadvantages are discussed. In Section 4.2.6, we propose a new space-discretization formulation that can handle all constraints discussed above. In Section 4.2.7, we propose a convex quadratic program-based method and a linear-time method to generate initial solutions. In Section 4.2.8, we propose a modified vertical cell decomposition method to generate convex constraints for dynamic obstacles. Experimental results are presented in Section 4.3, where three types of dynamic obstacles are tested: following (4.3.1), merging (4.3.2), and crossing (4.3.3).

4.2 Speed Planning along a Fixed Path

In this chapter, we address the speed planning problem for autonomous vehicles. We assume a path is already given, and only the speed profile along the path needs to be generated. The total travel time along the path is unknown and needs to be optimized. We also assume the future trajectories of dynamic obstacles are provided by a prediction module.

4.2.1 Path Representation and Discretization

The path is defined as $s(t)$: $[0, T] \rightarrow [0, L]$, where L is the total length of the path that is given and T is the unknown total travel time that needs to be optimized. The speed and accelerations along the path are denoted as $v(\cdot)$ and $a(\cdot)$, respectively.

The path can be discretized in two different ways based on time or space. Assuming there are N segments after the discretization, if it is discretized on time, then each segment has a uniform time gap of Δ_t, where $T = N\Delta_t$. The timestamps at each state are marked as $t = t_0, t_1, \ldots t_N$. Similarly, if it is discretized on position, then each segment has a uniform space gap of Δ_s, where $L = N\Delta_s$. The positions at each state are given by $s = s_0, s_1, \ldots s_N$. It is notable that generally $s(t_i)$ is not equal to s_i unless the ego vehicle travels at a constant speed. Similarly, $t(s_i)$ is not equal to t_i in general. Depending on which discretization is selected, very different optimization problems and characteristics result. Details are discussed in Section 4.2.4, 4.2.5, and 4.2.6.

4.2.2 Objective Functions

We consider three objectives in this chapter. These are the total travel time T, smoothness, and distance keeping with the leading vehicle.

Time

Under time discretization, T can be computed from Δ_t.

$$T = N\Delta_t \tag{4.1}$$

Under space discretization, T can be computed from Δ_s and $v(s_i)$.

$$T = \sum_{i=0}^{N-1} \frac{\Delta_s}{v(s_i)} \tag{4.2}$$

Smoothness

The smoothness measure, denoted as R, is defined as the squared sum of the rate of change of the acceleration.

$$R(\cdot) = \sum_{i=0}^{N-1} \left(a_{i+1}(\cdot) - a_i(\cdot)\right)^2 \tag{4.3}$$

Distance Keeping

Distance keeping means maintaining a safe distance between the ego vehicle and the leading vehicle. Next, we will discuss how to define the cost function for distance keeping under the two different discretizations.

For the time discretization, at time t_i, assume the leading vehicle travels at speed $v_l(t_i)$, and its position is $s_l(t_i)$. Assuming we want to maintain a time headway t_h, and a minimal safe distance d_{min} between the ego vehicle and the leading vehicle, then a desired position $s_d(t_i)$ that keeps a safe distance from the leading vehicle can be defined as

$$s_d(t_i) = s_l(t_i) - d_{min} - t_h v_l(t_i) \tag{4.4}$$

The distance keeping cost under time discretization is

$$D(t) = \sum_{i=0}^{N-1} (s(t_i) - s_d(t_i))^2 \tag{4.5}$$

For the space discretization, at position s_i, assume the leading vehicle travels at speed $v_l(s_i)$, and its timestamp is $t_l(s_i)$. Given the safe time headway t_h, a desired timestamp $t_d(s_i)$ at position s_i can be defined as

$$t_d(s_i) = t_l(s_i) + t_h \tag{4.6}$$

The distance keeping cost under space discretization is

$$D(s) = \sum_{i=0}^{N-1} (t(s_i) - t_d(s_i))^2 \tag{4.7}$$

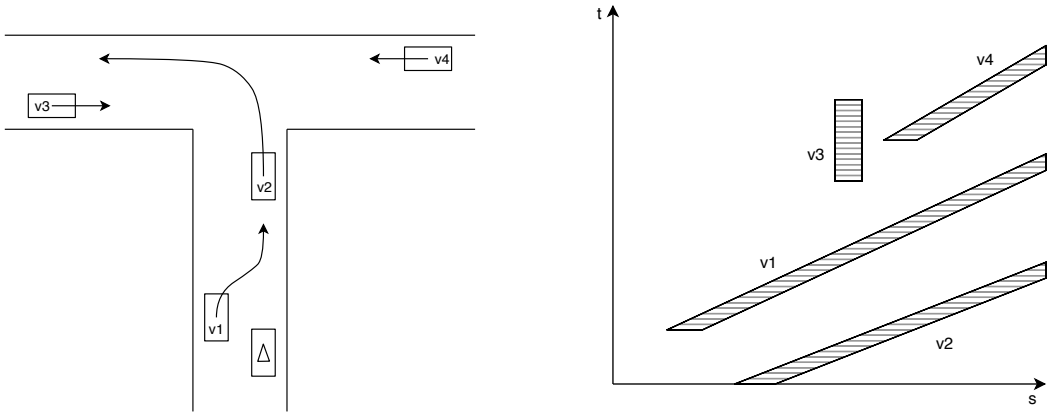
If there is no leading car, optionally, we can create a virtual leading car based on the minimum-time solution introduced in Section 4.2.7. The virtual leading car is the optimal solution when there are no obstacles, and it can help the optimization converge faster. In some merging situations, where the autonomous vehicle merges behind a car (e.g. v_1 and v_4 in Fig 4.1), the leading car only intersects with part of the autonomous vehicle's path. In this case, a hybrid leading car can be generated from a combination of the virtual leading car from the minimum-time solution and the real leading car. At each timestamp, whichever is closer to the autonomous vehicle is selected. The same rule applies to multiple leading vehicles.

4.2.3 Constraints

Three types of constraints are considered: dynamic obstacles, speed, and acceleration.

Dynamic Obstacle Representation in $s \times t$ Space

For dynamic obstacles, we can compute their intersections with the given path, which yields forbidden regions in the $s \times t$ space. Some typical urban driving scenarios are shown in Fig 4.1, which from the perspective of the ego vehicle marked with a triangle includes following a leading car (v_2), interacting with a cutting-in vehicle (v_1), merging into the main traffic (v_4) from the side road, and turning left with cross traffic (v_3).



(a) T-intersection. The ego vehicle marked with a triangle turns left at the intersection.

(b) Forbidden regions.

Figure 4.1: Dynamic obstacle representation.

Obviously, the forbidden regions formed by dynamic obstacles have non-convex shapes. We first apply a modified vertical cell decomposition method to generate collision-free channels in the $s \times t$ space, such that at each timestamp or

position dynamic obstacles can be expressed as convex constraints. Details of the modified vertical cell decomposition method are discussed in Section 4.2.8.

Under time discretization, the dynamic obstacle constraints can be expressed as

$$\underline{s}(t_i) \leq s(t_i) \leq \bar{s}(t_i) \tag{4.8}$$

Under space discretization, the dynamic obstacle constraints can be expressed as

$$\underline{t}(s_i) \leq t(s_i) \leq \bar{t}(s_i) \tag{4.9}$$

Speed Constraints

Speed constraints come from several different sources: the road speed limit, rollover prevention, and driving comfort. The last three are all related to lateral acceleration a_y, which can be defined as

$$a_y = \frac{v^2}{R(s)} = v^2 \kappa(s) \tag{4.10}$$

where R is the turning radius and κ is the curvature. The turning radius R and the curvature κ are known for a fixed path but vary at different positions on the path.

Road speed limit The road speed limit is position-dependent when the path crosses an intersection. For example, when the vehicle turns right/left onto the main road from a side road, the speed limits of the main road and the side road could be different. In this case, methods from ([64, 66]) cannot be directly applied since they assume a global speed limit. The constraints for the road speed limits

denoted as $v_r(s)$ can be defined below

$$v(\cdot) \leq \bar{v}_r(s) \tag{4.11}$$

Rollover prevention A simplified static condition for rollover to occur is ([88]):

$$a_y > \frac{g l_w}{2h} \tag{4.12}$$

where g is the acceleration due to gravity, l_w is the track width, and h is the height of the center of gravity of the vehicle. Combining (4.10) and (4.12) gives the speed condition to prevent rollover

$$v(\cdot) \leq \sqrt{\frac{g l_w}{2h\kappa(s)}} \tag{4.13}$$

Driving comfort To ensure driving comfort, small lateral accelerations are necessary conditions ([89]). The constraints on driving comfort can be expressed as

$$v(\cdot) \leq \sqrt{\frac{\bar{a}_y}{\kappa(s)}} \tag{4.14}$$

where $\bar{a}_y$ is the maximum lateral acceleration for comfort. Sometimes these hard constraints might be too strict, limiting the motion of the vehicle and causing the optimization to be infeasible. They can be replaced as soft constraints with slack variables:

$$\begin{aligned} v(\cdot) &\leq \sqrt{\frac{\bar{a}_y}{\kappa(s)}} + \varsigma_c, \\ \varsigma_c &\geq 0 \end{aligned} \tag{4.15}$$

where ς_c is the slack variable. Penalties for the slack variables also need to be added in the objective function in this case.

The constraints from road speed limit (4.11), rollover prevention (4.13), and driving comfort (4.14) are all position-dependent. They can be merged into unified constraints by taking the minimum value $\bar{v}(s)$ of the right side of the equations at each position s_i:

$$v(\cdot) \leq \bar{v}(s) \tag{4.16}$$

An example that combines the road speed limits and speed constraints from the lateral acceleration is shown in Fig 4.2, where the autonomous vehicle turns onto the main road from a side road as shown in Fig 4.1.

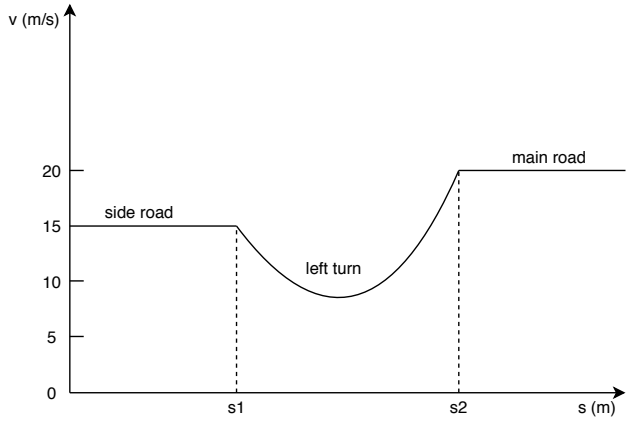


Figure 4.2: Speed limit

Acceleration Constraints

We simply assume constant acceleration constraints across the path, which are time- and space-independent.

$$\underline{a} \leq a(\cdot) \leq \bar{a} \tag{4.17}$$

4.2.4 Problem Formulation based on Time Discretization

The time-optimal speed planning problem discretized on time is formulated in (4.18). As discussed in Section 4.2.1, under time discretization, the total travel time T is evenly divided into N segments with each one having time gap Δ_t.

$$\min_{\Delta_t,\, s(\cdot),\, v(\cdot),\, a(\cdot)} \quad \alpha T + \beta R(t) + \gamma D(t) \tag{4.18a}$$

$$\text{s.t.} \quad s(t_{i+1}) = s(t_i) + v(t_i)\Delta_t, \tag{4.18b}$$

$$v(t_{i+1}) = v(t_i) + a(t_i)\Delta_t, \tag{4.18c}$$

$$s(t_0) = 0, \quad s(t_N) = L, \tag{4.18d}$$

$$v(t_0) = v_0, \quad v(t_N) = v_N, \tag{4.18e}$$

$$a(t_0) = a_0, \quad a(t_N) = a_N, \tag{4.18f}$$

$$\underline{s}(t_i) \le s(t_i) \le \bar{s}(t_i), \tag{4.18g}$$

$$0 \le v(t_i) \le \bar{v}(s(t_i)), \tag{4.18h}$$

$$\underline{a} \le a(t_i) \le \bar{a} \tag{4.18i}$$

The objective function (4.18a) is a weighted sum of the total travel time T (4.1), smoothness measure $R(t)$ (4.3), and cost of distance keeping $D(t)$ (4.5), where α, β, and γ are weighting parameters. The dynamic equations for position and speed are defined in (4.18b) and (4.18c) respectively. They are both nonlinear equations because Δ_t is not pre-determined, but an optimization variable. (4.18d)-(4.18f) are the boundary conditions for position, speed, and acceleration. (4.18g) is the dynamic obstacle constraint defined in (4.8). (4.18h) is the speed constraint defined in (4.16), which is position-dependent. (4.18i) is the acceleration constraint

defined in (4.17).

There are two factors that make Problem (4.18) non-convex and hard to solve. First, the dynamic equations (4.18b) and (4.18c) are nonlinear. Second, the speed constraint (4.18h) is position-dependent but the problem is discretized on time. Consequently, the speed constraints corresponding to the same timestamps will keep changing as the speed profile gets updated during iterations. The solution may oscillate before converging.

4.2.5 Problem Formulation based on Space Discretization

[68] proposes a new formulation based on space discretization. First, by fixing a space gap Δ_s and removing Δ_t from the optimization variables, (4.18b) and (4.18d) are no longer needed. As a result, $s(\cdot)$ and Δ_t can be removed from the optimization variables. Second, (4.18c) can be replaced with $v^2(s_{i+1}) - v^2(s_i) = 2a(s_i)\Delta_s$. Denoting $w = v^2$, this can be further written as

$$w(s_{i+1}) - w(s_i) = 2a(s_i)\Delta_s \tag{4.19}$$

The timestamp at each position s_i can also be computed from w and Δ_s

$$t(s_i) = \sum_{j=0}^{i-1} \frac{\Delta_s}{v(s_j)} = \sum_{j=0}^{i-1} \frac{\Delta_s}{\sqrt{w(s_j)}} \tag{4.20}$$

The total travel time T is

$$T = t(s_N) = \sum_{j=0}^{N-1} \frac{\Delta_s}{\sqrt{w(s_j)}} \tag{4.21}$$

The problem formulated using space discretization is defined in (4.22)

$$\min_{w(\cdot),\, a(\cdot)} \quad \alpha T + \beta R(s) + \gamma D(s) \tag{4.22a}$$

$$\text{s.t.} \quad w(s_{i+1}) - w(s_i) = 2a(s_i)\Delta_s, \tag{4.22b}$$

$$w(s_0) = v_0^2, \quad w(s_N) = v_N^2, \tag{4.22c}$$

$$a(s_0) = a_0, \quad a(s_N) = a_N, \tag{4.22d}$$

$$\underline{t}(s_i) \leq t(s_i) \leq \bar{t}(s_i), \tag{4.22e}$$

$$0 \leq w(s_i) \leq \bar{v}^2(s_i), \tag{4.22f}$$

$$\underline{a} \leq a(s_i) \leq \bar{a} \tag{4.22g}$$

The advantages of the space discretization formulation (4.22) over the time discretization formulation (4.18) are obvious. First, a linear dynamic equation (4.22b) replaces the nonlinear dynamic equations (4.18b) and (4.18c). Second, the speed and its constraints are both position-dependent. Therefore, the speed constraints (4.22f) also become linear constraints. The objective function (4.22a), defined as a weighted sum of the total travel time T (4.21), smoothness measure $R(s)$ (4.3), and the distance keeping cost $D(s)$ (4.7), is still a convex function. The problem (4.22) is a convex program if the dynamic obstacle constraints (4.22e) are not considered.

The only drawback is that the obstacle constraints (4.22e) become half convex and half non-convex under this formulation. That is, the right sides of the constraints are convex, but the left sides of the constraints are non-convex. If only the right sides of the constraints are imposed, the ego vehicle can only reach a position earlier than dynamic obstacles but not vice versa.

4.2.6 A New Space-Discretization Formulation

We propose a new formulation based on space discretization to better handle all four constraints. We denote the time gap between positions s_i and s_{i+1} as $\delta_t(s_i)$ and add them into the optimization variables. The timestamp at each position s_i can be represented as

$$t(s_i) = \sum_{j=0}^{i-1} \delta_t(s_j) \tag{4.23}$$

The total travel time T is a linear combination of $\delta_t(s_i)$

$$T = t(s_N) = \sum_{j=0}^{N-1} \delta_t(s_j) \tag{4.24}$$

The time-optimal speed planning problem (4.18) can then be reformulated as

$$\min_{\delta_t(\cdot), v(\cdot), a(\cdot)} \quad \alpha T + \beta R(s) + \gamma D(s) \tag{4.25a}$$

$$\text{s.t.} \quad v(s_i)\delta_t(s_i) = \Delta_s, \tag{4.25b}$$

$$a(s_i)\delta_t(s_i) = v(s_{i+1}) - v(s_i), \tag{4.25c}$$

$$v(s_0) = v_0, \quad v(s_N) = v_N, \tag{4.25d}$$

$$a(s_0) = a_0, \quad a(s_N) = a_N, \tag{4.25e}$$

$$\underline{t}(s_i) \le t(s_i) \le \bar{t}(s_i), \tag{4.25f}$$

$$0 \le v(s_i) \le \bar{v}(s_i), \tag{4.25g}$$

$$\underline{a} \le a(s_i) \le \bar{a} \tag{4.25h}$$

Under our formulation, the objective function (4.25a) is a convex function.

The dynamic obstacle (4.25f), speed (4.25g), and acceleration (4.25h) constraints are all convex constraints. However, the system dynamic equations (4.25b) and (4.25c) remain nonlinear functions. Next, we will show how to handle these nonlinear equality constraints efficiently.

The right sides of (4.25b) and (4.25c) are linear functions and the left sides are nonlinear functions. Denoting the left side of (4.25b) as $f(v(s_i), \delta_t(s_i))$ and taking the first-order Taylor expansion at (v_*, δ_*), we obtain

$$\begin{aligned} f(v(s_i), \delta_t(s_i)) \approx & f(v_*, \delta_*) + f_v(v_*, \delta_*)(v(s_i) - v_*) \\ & + f_{\delta_t}(v_*, \delta_*)(\delta_t(s_i) - \delta_*) \\ = & v_*\delta_* + \delta_* (v(s_i) - v_*) + v_*(\delta_t(s_i) - \delta_*) \\ = & \delta_* v(s_i) + v_*\delta_t(s_i) - v_*\delta_* \end{aligned} \tag{4.26}$$

Similarly, writing the left side of (4.25c) as $g(a(s_i), \delta_t(s_i))$, and taking the first-order Taylor expansion at (a_*, δ_*), we get

$$g(a(s_i), \delta_t(s_i)) \approx \delta_* a(s_i) + a_*\delta_t(s_i) - a_*\delta_* \tag{4.27}$$

With (4.26) and (4.27), we can design an iterative scheme to solve problem (4.25). Before each iteration, the values of (v_*, a_*, δ_*) are updated with the values from the last iteration except for the initial iteration. Algorithms to generate the initial solution are discussed in Section 4.2.7. The nonlinear dynamics equations (4.25b) and (4.25c) are replaced with their linear Taylor approximations (4.26) and (4.27) during each iteration. (4.25) becomes a convex quadratic program at each iteration with this modification.

To improve the stability of convergence and avoid invalid solutions during

optimization, we also introduce slack variables in (4.25b) and (4.25c). The dynamic equations with Taylor approximation and slack variables can be rewritten as

$$\delta_* v(s_i) + v_* \delta_t(s_i) - v_* \delta_* = \Delta_s + \varsigma_s(s_i) \tag{4.28}$$

$$\delta_* a(s_i) + a_* \delta_t(s_i) - a_* \delta_* = v(s_{i+1}) - v(s_i) + \varsigma_v(s_i) \tag{4.29}$$

where $\varsigma_s(\cdot)$ and $\varsigma_v(\cdot)$ are slack variables. The penalties for the slack variables are defined as

$$K(s) = \sum_{i=0}^{N-1} \left(\varsigma_s^2(s_i) + \varsigma_v^2(s_i)\right) \tag{4.30}$$

Problem (4.25) can then be rewritten as

$$\min_{\substack{\delta_t(\cdot), v(\cdot), a(\cdot), \\ \varsigma_v(\cdot), \varsigma_a(\cdot)}} \quad \alpha T + \beta R(s) + \gamma D(s) + \eta K(s) \tag{4.31a}$$

$$\text{s.t.} \quad \delta_* v(s_i) + v_* \delta_t(s_i) - v_* \delta_*,$$

$$= \Delta_s + \varsigma_s, \tag{4.31b}$$

$$\delta_* a(s_i) + a_* \delta_t(s_i) - a_* \delta_*,$$

$$= v(s_{i+1}) - v(s_i) + \varsigma_v, \tag{4.31c}$$

$$v(s_0) = v_0, \quad v(s_N) = v_N, \tag{4.31d}$$

$$a(s_0) = a_0, \quad a(s_N) = a_N, \tag{4.31e}$$

$$\underline{t}(s_i) \leq t(s_i) \leq \bar{t}(s_i), \tag{4.31f}$$

$$0 \leq v(s_i) \leq \bar{v}(s_i), \tag{4.31g}$$

$$\underline{a} \leq a(s_i) \leq \bar{a} \tag{4.31h}$$

Problem (4.31) is a convex quadratic program. It can be solved reliably and efficiently by QP solvers in polynomial time (e.g. OSQP [76], CVXPY [90]), where

Algorithm 1 Minimal time solution over a fixed path given acceleration limits

for each station s in forward order **do**
  if $v_{i+1}^2 - v_i^2 \geq 2\bar{a}\Delta_s$ **then**
    $v_{i+1} \leftarrow \sqrt{v_i^2 + 2\bar{a}\Delta_s}$
  end if
end for
for each station s in backward order **do**
  if $v_{i+1}^2 - v_i^2 \geq 2\underline{a}\Delta_s$ **then**
    $v_i \leftarrow \sqrt{v_{i+1}^2 - 2\underline{a}\Delta_s}$
  end if
end for

the global optimal solutions are guaranteed as well.

4.2.7 Initial Solution for the Speed Planning Problem

To generate an initial solution for the speed planning problem, we simplify the problem by removing the dynamic obstacle constraints and distance keeping cost. The best formulation for this simplified problem is the space discretization formulation defined in (4.22). As discussed in Section 4.2.5, (4.22) is a convex program without the dynamic obstacle constraints. The simplified problem can be defined as

$$\min_{w(\cdot),\, a(\cdot)} \quad \alpha T + \beta R(s) \tag{4.32a}$$

$$\text{s.t.} \quad w(s_{i+1}) - w(s_i) = 2a(s_i)\Delta_s, \tag{4.32b}$$

$$w(s_0) = v_0^2, \quad w(s_N) = v_N^2, \tag{4.32c}$$

$$0 \leq w(s_i) \leq \bar{v}^2(s_i), \tag{4.32d}$$

$$\underline{a} \leq a(s_i) \leq \bar{a} \tag{4.32e}$$

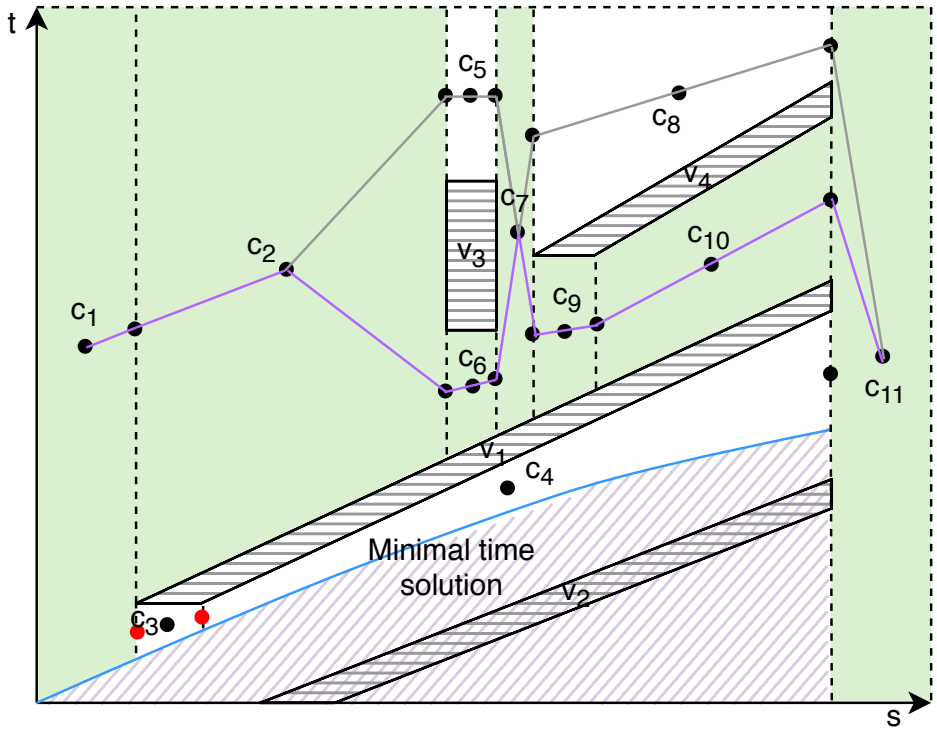


Figure 4.3: Vertical cell decomposition for the scenario in Fig. 4.1. An example of a feasible path is shown as a purple line, and its corresponding channel is filled with green.

(4.32) has a convex objective function, affine equality and inequality constraints. Therefore, it is a convex quadratic program and can be solved in polynomial time.

If we limit the acceleration commands to only maximum acceleration or deceleration, this problem can be solved in linear time based on a two-pass method. The algorithm is presented in Alg. 1.

4.2.8 Handling Dynamic Obstacles with Cell Decomposition

The constraints from dynamic obstacles in $(s \times t)$ space are non-convex in

Algorithm 2 The modified vertical cell decomposition algorithm

1: Generate forbidden regions in $(s \times t)$ space based on intersections among dynamic obstacles and the given path.
2: Compute the minimum-time solution (the blue line in Fig. 4.3) based on Alg. 1 and use it as the lower bound. Mark any regions below the minimum-time solution as forbidden regions.
3: Draw upper and lower vertical lines for each obstacle vertex until reaching another edge of the environment.
4: Place a node at the center of each trapezoid (cell) and the center of each upper and lower vertical line except for the ones without enough time gap (marked with red dots).
5: Build a connectivity graph based on adjacency.
6: Find all paths from the source node to the destination using breadth-first search (BFS). Keep track of the maximum value of the lower bound of visited cells during expansion. A path will stop expansion if the upper bound of a successor cell is lower than the maximum value of the lower bound of visited cells. For example, there is no feasible path between c_5 and c_9 because the lower bound of c_5 is higher than the upper bound of c_9.
7: Each feasible path corresponds to a channel of the freespace. The convex constraints are defined by the lower and upper bound of cells along the path.

general. We adapt a vertical cell decomposition method ([91]) to convert dynamic obstacles into convex constraints. Vertical cell decomposition is also called trapezoid decomposition. Its idea is to divide the freespace into small polygons called cells using vertical lines. The cells are either trapezoids or triangles, which are both convex shapes. After the decomposition, a connectivity graph is generated based on the adjacency of the cells, where each node of the graph corresponds to a cell and each edge of the graph connects two nodes of adjacent cells. Each solution of the connectivity graph between the start and the goal nodes represents a homotopy of free paths, which is called a channel or corridor of the freespace. The constraints defined by the channels are guaranteed to be convex constraints because each cell along the path has a convex shape. The vertical cell decomposition method has $O(n \log n)$ computational complexity with plane-sweep or line-sweep algorithms [92]

from computational geometry, where n is the number of vertices in the graph.

To apply the vertical cell decomposition method to the speed planning problem in $(s \times t)$ space, some additional heuristics can be exploited to further reduce the running time. First, any feasible solution should be no faster than the minimum-time solution produced by Alg. 1 from Section 4.2.7. Second, the time gap between two obstacles should be big enough. Third, time should be monotonically increasing.

The modified vertical cell decomposition algorithm is summarized in Alg. 2. An example is illustrated in Fig. 4.3. There might be multiple feasible channels produced by the algorithm. For each feasible channel, we obtain a Problem (4.31) with different dynamic obstacle constraints. In this case, different strategies can be used. We can solve all problems in parallel and choose the one with the best cost, or we can solve them sequentially until we get a valid solution. For the sequential strategy, the order can be determined based on which channel can provide the potential minimal time or other heuristics.

4.3 Experimental Results

We test the proposed method in the scenario shown in Fig. 4.1 with different types of dynamic obstacles, where the autonomous vehicle makes a left turn from a side road onto a main road. It interacts with four types of dynamic obstacles: following a leading car with varying road speed limit (v_2), handling a cut-in car (v_1), merging into the main traffic (v_4), and turning left with cross traffic (v_3). We conduct testing on three of them given that the cut-in and merging cases are very similar. In each test, only one dynamic obstacle appears in the scenario for better comparison. For each case, 100 different initial conditions are evaluated. Some typical results are discussed in Section 4.3.1, 4.3.2, 4.3.3 and 4.3.4. Two videos that show more

complicated scenarios can be found in https://youtu.be/9EDLI9X7HHc and https://youtu.be/RWxZbyagrBE. Parameters of the optimization problem (4.31) are listed in Table 4.1.

Table 4.1: Parameters.

N	40	α	20
t_h	1.5 s	β	1
$\bar{a}$	3 m/s^2	γ	5
$\underline{a}$	-5 m/s^2	ξ	100

The algorithm is implemented in Python with CVXPY [90], and the optimization problem (4.31) is solved using OSQP [76] with warm start enabled. We choose OSQP over interior point methods to solve the quadratic program because interior point methods do not benefit from warm start and our algorithm takes several iterations to satisfy the nonlinear system dynamic equations. The tests are implemented on a Desktop PC with an Intel i7-9700K CPU (8 core 3.60GHz) and 32GB memory. Average runtime and number of iterations are summarized in Table 4.2. It is able to find optimal solutions in a few iterations and on a time scale that is feasible for realtime autonomous driving applications. As the optimization approaches convergence, the latter iterations take less than 1 ms thanks to the warm start.

We compare our method with the formulation (4.22) proposed in [68]. There are generally two ways to solve nonlinear problems like (4.22): sequential quadratic programming (SQP) or interior point methods (IPM). We implemented both methods using CasADi [93]. For SQP, we use OSQP as the QP solver. For IPM, we use IPOPT [82]. The results are summarized in Table 4.2. [68] with SQP failed to find solutions for around one third of the tests. [68] with IPM can produce almost the same solutions but with much longer computational time.

Table 4.2: Average runtime in milliseconds (ms). Number of iterations of our method are included in brackets.

Scenarios	Ours	[68] with IPM	[68] with SQP
Case 1: Follow	2.65 (3)	23.58	39.20
Case 2: Merge	4.44 (5)	31.67	34% failed
Case 3: Cross (yield)	7.06 (5)	33.26	100% failed
Case 4: Cross (proceed)	2.40 (2)	24.93	62.39

4.3.1 Following a Leading Car with Varying Road Speed Limit

In this test, the autonomous vehicle follows a leading car (v_2 in Fig. 4.1) while making a left turn. The speed limit varies at different positions along the path. The leading car drives at 8 m/s, and the initial distance is 15 m. This test demonstrates our method can simultaneously handle position-dependent speed limit and dynamic obstacle constraints in the $s \times t$ space. The results are shown in Fig. 4.4. The top subplot shows the speed limit, the minimum-time solution, and the optimal speed at each position. The middle subplot draws the obstacles in the $s \times t$ space, the channel corresponding to the freespace, and the optimal solution represented in $s \times t$ space. The lower bound of the channel is formed by the upper side of the obstacle and the minimum-time solution. The optimal solution tracks the safe headway (4.6) that appears in the objective function, which is the desired time gap we want to maintain. The bottom subplot displays the acceleration profiles of the minimum-time solution and the optimal solution respectively.

4.3.2 Merging into the Main Traffic

In the second test, the autonomous vehicle makes a left turn and merges onto the main road. It yields to the car (v_4 in Fig. 4.1) on the main road and follows it. The

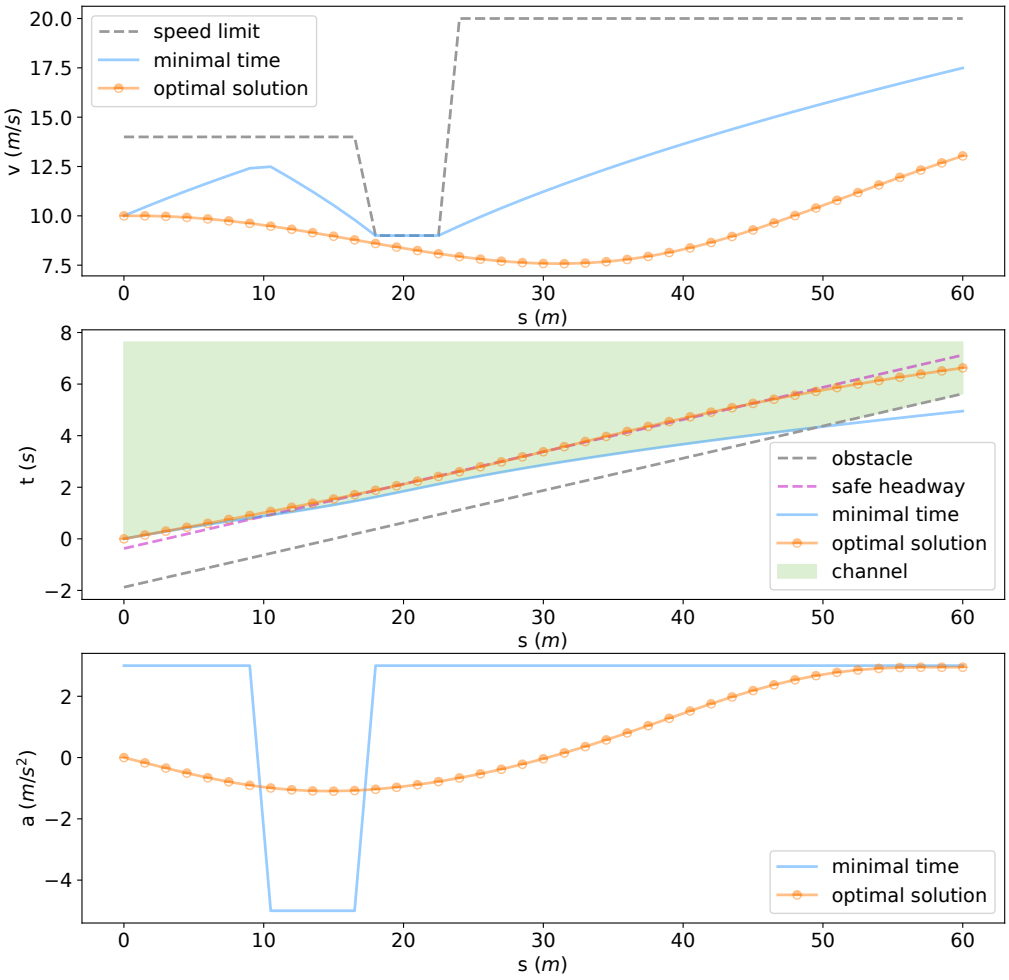


Figure 4.4: Case 1: following a leading car with varying road speed limit.

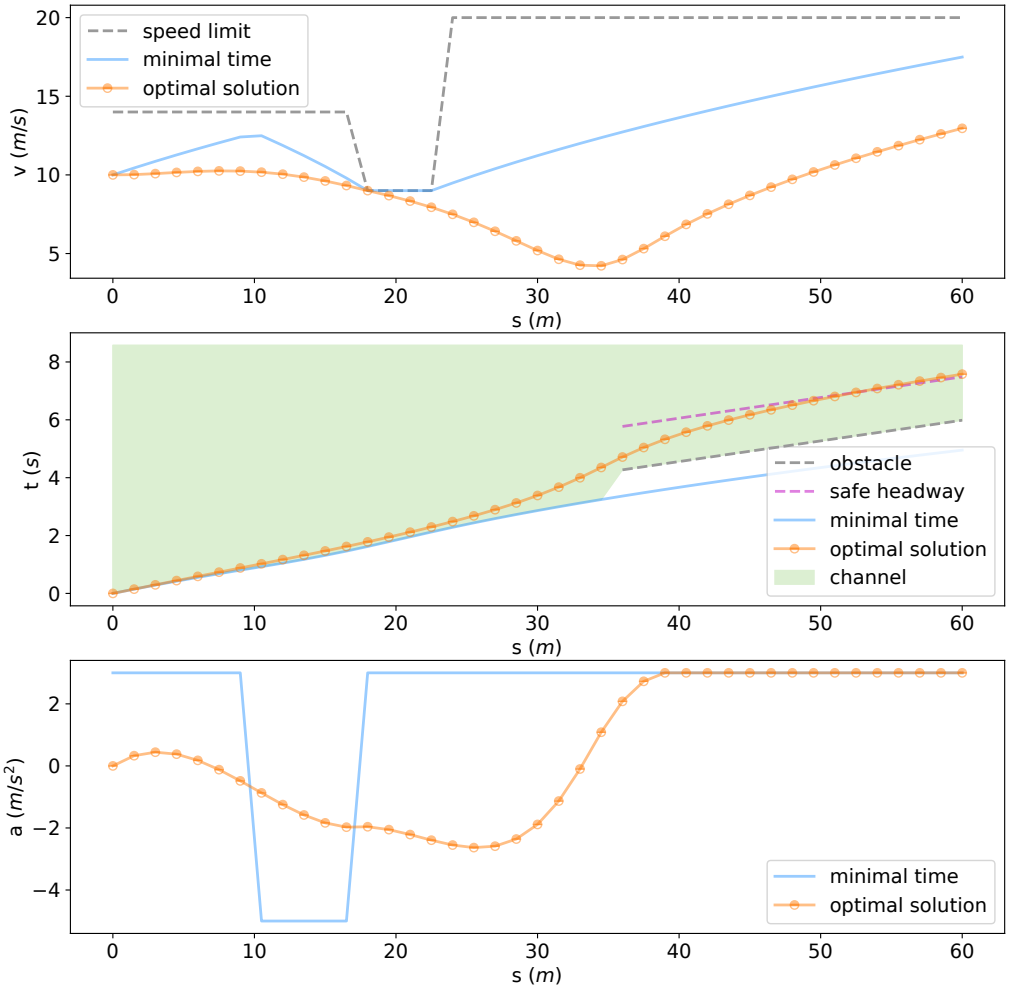


Figure 4.5: Case 2: merging behind a car on the main road.

results are shown in Fig. 4.5. In the top subplot, the optimal solution keeps under the speed limit, satisfying the speed limit constraints. In the middle subplot, the optimal solution stays away from the obstacle, denoted by the dashed gray line, and gradually converges to the safe headway denoted by the dashed purple line. The acceleration of the optimal solution is smooth, as shown in the bottom subplot.

4.3.3 Making an Unprotected Left Turn with Cross Traffic

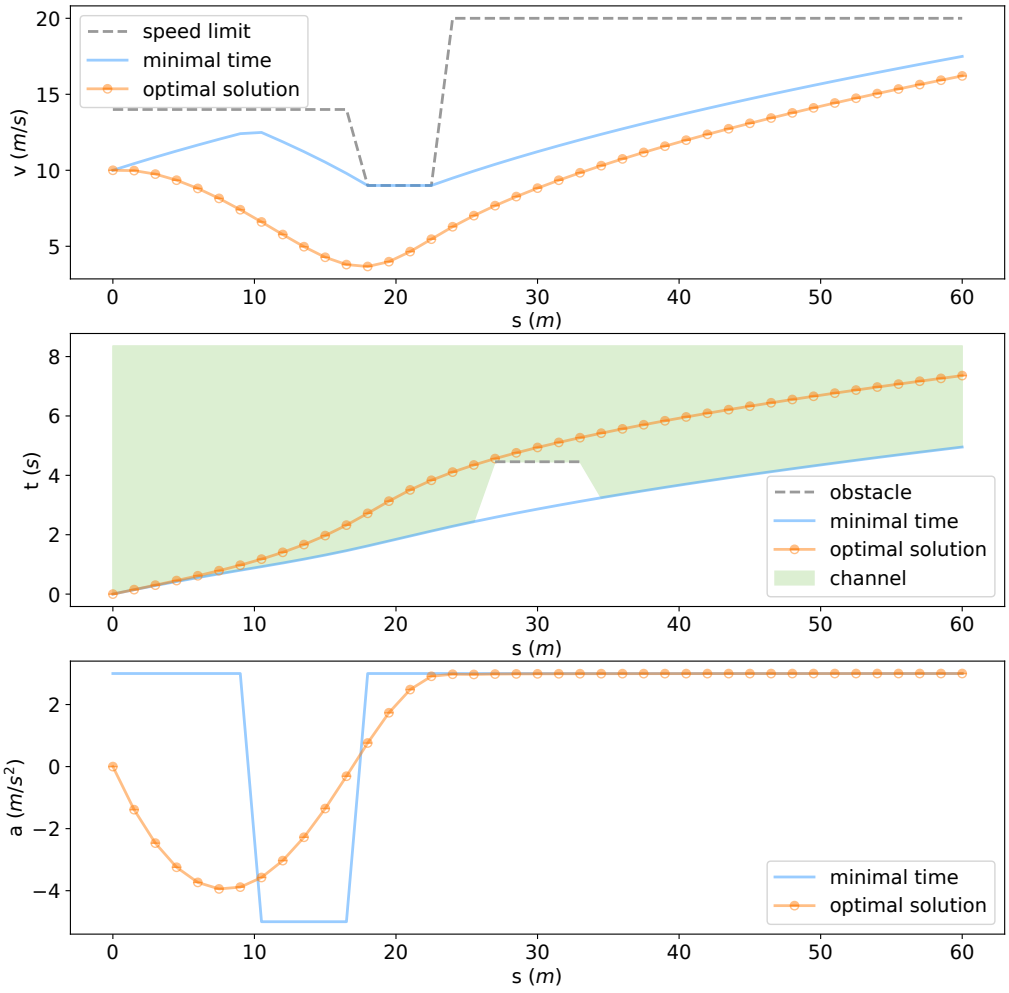


Figure 4.6: Case 3: yielding to cross traffic during a left turn.

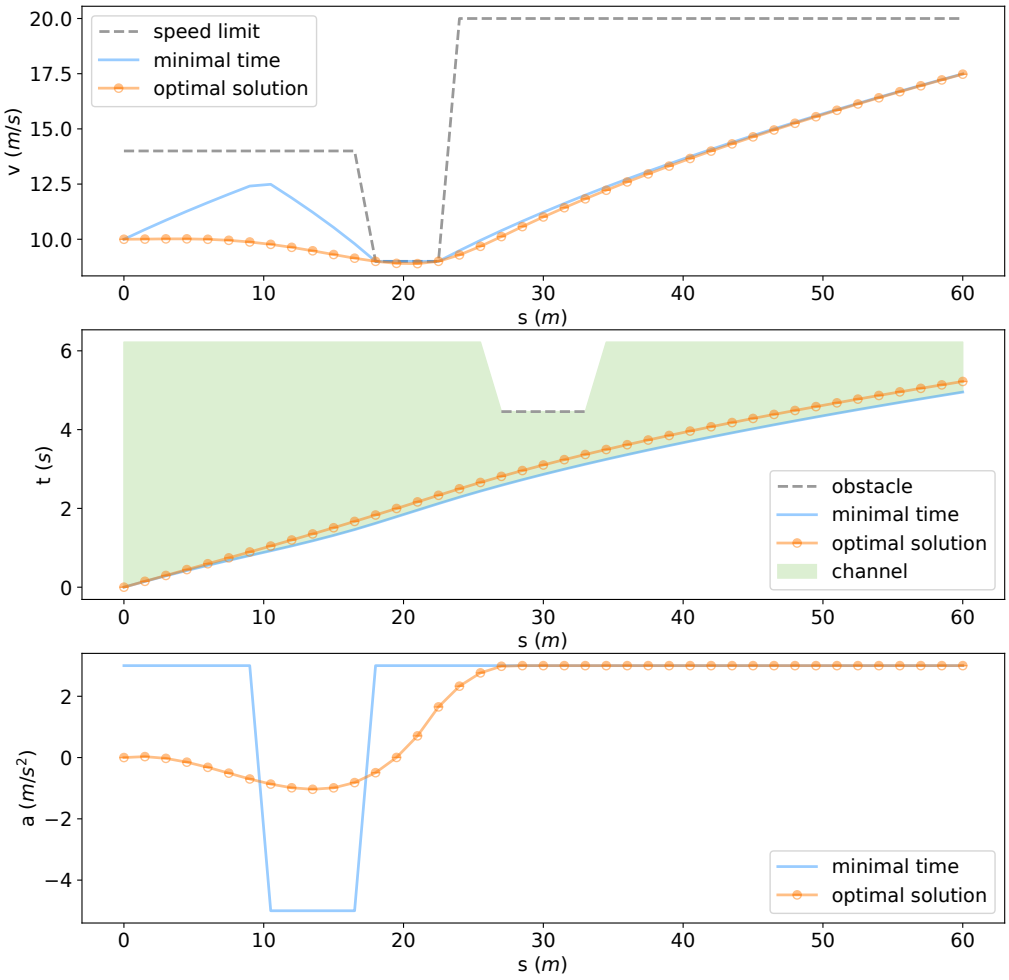


Figure 4.7: Case 4: proceeding ahead of the cross traffic during a left turn.

In the third test, the autonomous vehicle makes an unprotected left turn and encounters cross traffic (v_3 in Fig. 4.1). The obstacle divides the freespace into 2 homotopies (see middle subplots of Fig. 4.6 and Fig. 4.7). In the first case, the autonomous vehicle yields to the cross traffic (Fig. 4.6). It slows down to let the obstacle crosses first, and then accelerates to minimize the travel time. In the second case, the autonomous vehicle proceeds ahead of the cross traffic (Fig. 4.7). Because it needs to arrive at the collision point earlier than the obstacle, it almost follows the minimum-time solution.

4.3.4 Unprotected Left Turn with Both Cross Traffic and Merging Traffic

In this test, we combine Case 2 and Case 3, where the autonomous vehicle needs to yield to the cross traffic and then merge behind a car on the main road (v_3 and v_4 in Fig. 4.1). As we can see in Fig. 4.8, it slows down to let v_3 cross first, and then accelerates to merge behind v_4. In the middle subplot of Fig. 4.8, the optimal solution stays away from both obstacles, and follows the safe headway closely.

4.4 Conclusions

In this chapter, we presented an iterative convex optimization method for speed planning over a fixed path for autonomous driving. Our formulation optimizes multiple objectives together, such as time and smoothness minimization and distance keeping with the leading car, while obeying several constraints, such as the monotonicity time constrains, system dynamics constraints, dynamic obstacle constraints, and speed and acceleration constraints. We also presented a modified vertical cell decomposition method to handle dynamic obstacles. Finally, we apply this method

to typical autonomous driving scenarios such as following, merging, and crossing. It is able to find optimal solutions in a few iterations and on a time scale that is feasible for realtime autonomous driving applications.

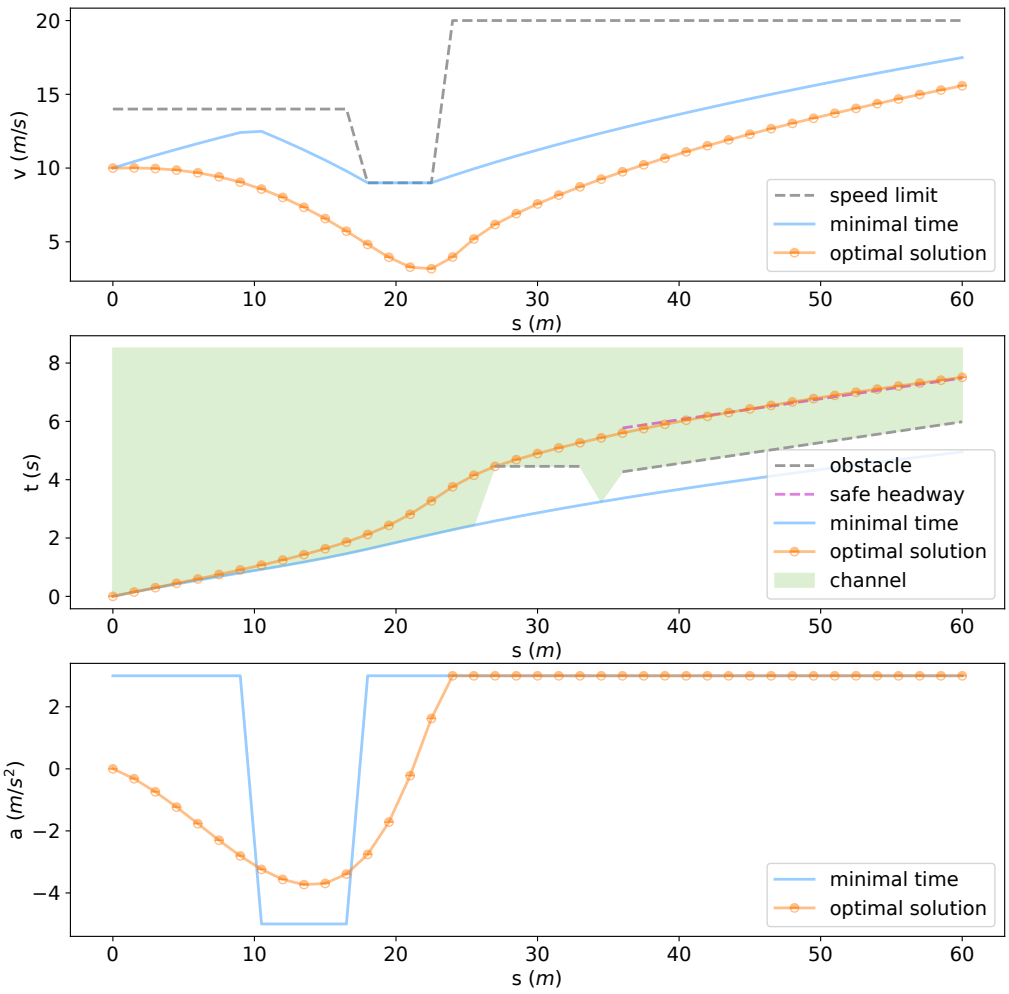


Figure 4.8: Case 5: yielding to cross traffic and merge behind a car on the main road.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this dissertation, we presented a sequential optimization approach for autonomous vehicle motion planning. In Chapter 3, we proposed an efficient sequential convex optimization approach for path planning with static obstacles. We converted the nonlinear kinematic model to a linear spline model based on differential flatness theory. Our method also allows imposing constraints on curvature. We further reformulated the spline model to a recurrent formulation, enabling a more efficient way to solve the optimization problem. Our method reveals a strong relationship among smoothing spline, multiple-shooting-based trajectory optimization, and iterative LQR (iLQR) or differential dynamic programming (DDP). Compared to the commonly used iLQR and DDP methods, our method allows imposing hard constraints, does not require warm start, and reduces computational time. In Chapter 4, we proposed a novel optimization formulation for speed planning along a fixed path with dynamic obstacles. Our method can handle point-wise speed constraint and dynamic obstacle constraints at the same time. To handle the dynamic obstacles,

we adapted a modified vertical cell deposition method to generate convex channels in the space-time ($s \times t$) space, which converts the dynamic obstacles into convex constraints. We also use the minimal-time solution and other heuristics to further accelerate the process.

Our work provides an efficient way to handle motion planning problems for autonomous vehicles. It can be used separately to solve a path planning problem or a speed planning problem along a fixed path, or can be combined to become a standalone module in a behavior planning framework. For example, we can generate paths for different behaviors (for instance, lane follow and lane change), and plan optimal speed profiles along each of them.

Our work assumes that the autonomous vehicle drives under normal conditions. In cases where an evasive maneuver is needed, a high-fidelity dynamic model is needed to better predict the motion of the vehicle instead of the kinematic model used in this work. Some preliminary work that uses a dynamic bicycle model in evasive situations is presented in Section 5.2.1. Our work also assumes that the predicted trajectories of other objects' motion are given before the planning is started. In cases where there are interactions between the autonomous vehicle and other objects, such as lane change in congested traffic, such assumptions do not hold. In such cases, combining planning and prediction together might be a better option. More discussions are presented in Section 5.2.2 and 5.2.3.

5.2 Future Work and Open Questions

Many future directions may be explored. The first one is to improve the trajectory using a dynamic bicycle model to handle evasive situations. The second one is to extend the presented method to handle more scenarios, such as lane change.

5.2.1 Motion Planning with Dynamic Bicycle Model

One possible future direction is to refine the trajectory using the dynamic bicycle model. Compared to the kinematic model, the dynamic bicycle model considers the side-slip motion of the vehicle. It can better describe the vehicle's motion, especially at high speed. Next, we present a conventional way to solve this problem.

Dynamic Bicycle Model

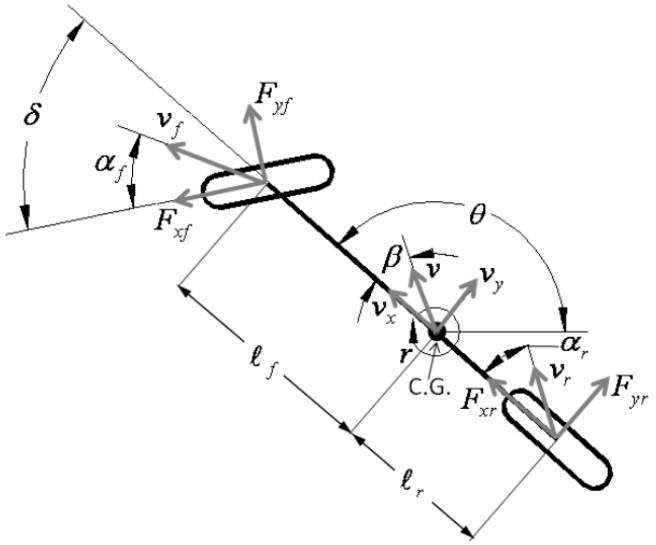


Figure 5.1: Dynamic bicycle model

A dynamic bicycle model is shown in Fig. 5.1. To model the side slip of the vehicle, a seven-degree-of-freedom dynamic model is considered. The state of the vehicle is defined as $\mathbf{x} = (x, y, \theta, v_x, v_y, r, \delta)$, where (x, y) is the 2D location of the center of gravity of the vehicle, θ is the heading, v_x is the longitudinal velocity along the heading, v_y is the lateral velocity, r is the yaw rate, and δ is the front ground wheel angle. The control variables are $\mathbf{u} = (u_a, u_\delta)$, where u_a is the acceleration,

and u_δ is the turning rate of the front ground wheel.

Considering the lateral dynamics of the vehicle by balancing lateral forces and yaw moment gives

$$\begin{aligned} m\left(\dot{v}_y + v_x r\right) &= F_{yf}\cos(\delta) - F_{xf}\sin(\delta) + F_{yr} \\ I_z\dot{r} &= \ell_f\left(F_{yf}\cos(\delta) - F_{xf}\sin(\delta)\right) - \ell_r F_{yr} \end{aligned} \tag{5.1}$$

where m is the vehicle mass, and r is the angular velocity about the yaw axis. ℓ_f and ℓ_r are the distance from the center of gravity to the front and rear wheel, respectively. The slip angles of the tires are given by

$$\begin{aligned} \alpha_f &= \tan^{-1}\left(\frac{v_y + \ell_f r}{v_x}\right) - \delta \\ \alpha_r &= \tan^{-1}\left(\frac{v_y - \ell_r r}{v_x}\right) \end{aligned} \tag{5.2}$$

Modeling the force generated by the wheels as linearly proportional to the slip angle, the lateral forces are defined as

$$\begin{aligned} F_{yf} &= -c_f\alpha_f \\ F_{yr} &= -c_r\alpha_r \end{aligned} \tag{5.3}$$

Substituting Eqn. 5.2 and Eqn. 5.3 into Eqn. 5.1 yields

$$\begin{bmatrix} \dot{v}_y \\ \dot{r} \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} v_y \\ r \end{bmatrix} + \begin{bmatrix} E \\ F \end{bmatrix}\delta \tag{5.4}$$

where A, B, C, D, E, F stand for

$$\begin{aligned} A &= -\frac{c_f + c_r}{m v_x} \\ B &= \frac{-\ell_f c_f + \ell_r c_r}{m v_x} - v_x \\ C &= \frac{-\ell_f c_f + \ell_r c_r}{I_z v_x} \\ D &= -\frac{\ell_f^2 c_f + \ell_r^2 c_r}{I_z v_x} \\ E &= \frac{c_f}{m} \\ F &= \frac{\ell_f c_f}{I_z} \end{aligned} \tag{5.5}$$

Combining everything together gives the full dynamic bicycle model $\dot{\mathbf{x}} = f_d(\mathbf{x}, \mathbf{u})$

$$\begin{aligned} \dot{X} &= v_x \cos(\theta) - v_y \sin(\theta) \\ \dot{Y} &= v_x \sin(\theta) + v_y \cos(\theta) \\ \dot{\theta} &= r \\ \dot{v_x} &= u_a \\ \dot{v_y} &= A v_y + B r + E \delta \\ \dot{r} &= C v_y + D r + F \delta \\ \dot{\delta} &= u_\delta \end{aligned} \tag{5.6}$$

Trajectory Optimization with Dynamic Model

Our goal is to refine the kinematic trajectory produced by our motion planning approach with the dynamic bicycle model. The state of the kinematic trajectory is defined as $\hat{\mathbf{x}} = \left(\hat{x}, \hat{y}, \hat{\theta}, \hat{v_x}, \hat{v_y}, \hat{r}, \hat{\delta}\right)$. The time span of the kinematic trajectory is T. We fix the speed along the trajectory and only optimize the steering com-

mand. Therefore, the time span T remains the same in the new optimization, and the dynamic obstacle constraints are automatically satisfied. We define the lateral distance d between the corresponding states of the two trajectories.

$$d_{lat} = -(x - \hat{x}) \sin \hat{\theta} + (y - \hat{y}) \cos \hat{\theta} \tag{5.7}$$

The state of the new trajectory is defined as $\mathbf{x} = (x, y, \theta, v_y, r, \delta)$, the parameter $\mathbf{p}$ is the longitudinal velocity $\hat{v_x}$, and the control variable $\mathbf{u}$ is the rate of ground steering angle $\dot{\delta}$. The resulting optimization problem is formulated as follows

$$\begin{aligned}
\underset{\mathbf{x},\mathbf{u}}{\text{minimize}} \quad & \int_0^T \left(\lambda d^2 + \mathbf{u}^T R \mathbf{u}\right) dt && \text{(5.8a)} \\
\text{subject to} \quad & \dot{\mathbf{x}} = f_d(\mathbf{x}, \mathbf{u}, \mathbf{p}), && \text{(5.8b)} \\
& -\Delta_l \leq d_{lat} \leq \Delta_r, && \text{(5.8c)} \\
& \mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max}, && \text{(5.8d)} \\
& \mathbf{u}_{\min} \leq \mathbf{u} \leq \mathbf{u}_{\max}, && \text{(5.8e)} \\
& \mathbf{x}_0 = \mathbf{x}_{init}, && \text{(5.8f)} \\
& \mathbf{x}_N = \mathbf{x}_{goal} && \text{(5.8g)}
\end{aligned}$$

We can solve (5.8) using a multiple-shooting trajectory optimization method [48]. The key idea of multiple shooting is to discretize time T into N intervals, where the time points are denoted as $(t_0, t_1, \ldots, t_{N-1})$. The length of each interval is denoted as h. Within each interval, the states are obtained by propagating the differential equation (5.8b). Constraints are then added to ensure the intervals join at boundaries. We can use the Runge-Kutta method to integrate the differential

equation. The differential equation constraints (5.8b) are replaced by the boundary constraints.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + g_d(\mathbf{x}_k, \mathbf{u}_k, \mathbf{p}_k) \tag{5.9}$$

where

$$g_d(\mathbf{x}_k, \mathbf{u}_k, \mathbf{p}_k) = \frac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k4\right) \tag{5.10}$$

$$k_1 = f_d(\mathbf{x}_k, \mathbf{u}_k, \mathbf{p}_k) \tag{5.11}$$

$$k_2 = f_d(\mathbf{x}_k + \frac{1}{2}k_1 h, \mathbf{u}_c, \mathbf{p}_c) \tag{5.12}$$

$$k_3 = f_d(\mathbf{x}_k + \frac{1}{2}k_2 h, \mathbf{u}_c, \mathbf{p}_c) \tag{5.13}$$

$$k_4 = f_d(\mathbf{x}_k + k_3 h, \mathbf{u}_{k+1}, \mathbf{p}_{k+1}) \tag{5.14}$$

$$\mathbf{u}_c = \mathbf{u}(t_k + \frac{h}{2}), \quad \mathbf{p}_c = \mathbf{p}(t_k + \frac{h}{2}) \tag{5.15}$$

With the boundary constraints (5.9), the optimal control problem (5.8) can be transcribed as a nonlinear program (NLP) below. This NLP (5.16) can be solved by sequential quadratic programming (SQP) or interior point methods (IPM).

$$\underset{\mathbf{x}_k,\,\mathbf{u}_k}{\text{minimize}} \quad \sum_{k=0}^{N-1} \left(\lambda d_k^2 + \mathbf{u}_k^T R \mathbf{u}_k\right) \tag{5.16a}$$

$$\text{subject to} \quad \mathbf{x}_{k+1} = \mathbf{x}_k + g_d(\mathbf{x}_k, \mathbf{u}_k, \mathbf{p}_k), \tag{5.16b}$$

$$-\Delta_l \leq d_k \leq \Delta_r, \tag{5.16c}$$

$$\mathbf{x}_{\min} \leq \mathbf{x_k} \leq \mathbf{x}_{\max}, \tag{5.16d}$$

$$\mathbf{u}_{\min} \leq \mathbf{u_k} \leq \mathbf{u}_{\max}, \tag{5.16e}$$

$$\mathbf{x}_0 = \mathbf{x}_{init}, \tag{5.16f}$$

$$\mathbf{x}_N = \mathbf{x}_{goal} \tag{5.16g}$$

Possible Improvements

In this subsection, so far we have presented a typical multiple-shooting-based trajectory optimization approach to solve Problem (5.8). This is a nonlinear and non-convex optimization problem, which in general is hard and time-consuming to solve. One possible idea is to express the dynamic bicycle model in path coordinates. In this way, the lateral dynamics may be converted to a linear model with respect to the kinematic trajectory. Therefore, we may be able to reformulate (5.8) as a quadratic program (QP).

5.2.2 Lane Changes

Our motion planning approach can handle many urban driving scenarios for autonomous vesicles. For example, following a leading car while biasing away from static obstacles, handling cut-in vehicles, merging into the main traffic, making unprotected turns, interacting with pedestrians on crosswalk. However, when and how

to make lane changes are still challenging, especially in dense traffic. The first question is when to make lane changes. There are many factors that affect lane change decisions. A vehicle may make a lane change due to routing needs, overtaking a slow moving vehicle, or road blockage. The second question is how to make lane changes. The key question here is how to choose an endpoint in the destination lane. Once the endpoint is determined, we can generate a path using the path optimization method presented in Chapter 3, and then generate a speed profile along the path using the speed optimization method presented in Chapter 4.

5.2.3 Open Questions

There are still many open questions to address for motion planning in urban scenarios. To reduce the computational cost, a planning system for an autonomous vehicle normally decomposes into route planning, behavior planning, and motion planning. If we make early decisions in the upper layers, then flexibility of maneuvers in the lower layers may be limited. How to find a good balance among these hierarchical layers is still an open question. Another open question lies in the relationship between prediction and motion planning. A commonly used assumption is that the planning module knows other traffic participants' motion beforehand. However, in many crowd scenarios there are interactions among the host vehicle and the traffic participants. Therefore, such an assumption may not hold, since the host vehicle's decision will affect traffic participants' future motion. A better way might be embedding the prediction system into the planning module. A third open question is how to handle uncertainties from the perception or localization system in motion planning. A preliminary examination of this problem has been performed in one of our previous works [94].

Bibliography

[1] I. U.S. Census Bureau, Chicago, "Motor vehicle accidents-number and deaths: 1990 to 2009." http://www.census.gov/compendia/statab/2012/tables/12s1103.pdf, 2012. [Online; accessed 15-September-2015]. 1.1

[2] C. Urmson, J. Anhalt, J. A. D. Bagnell, C. R. Baker , R. E. Bittner, J. M. Dolan, D. Duggins, D. Ferguson , T. Galatali, H. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. Howard, A. Kelly, D. Kohanbash, M. Likhachev, N. Miller, K. Peterson, R. Rajkumar, P. Rybski, B. Salesky, S. Scherer, Y.-W. Seo, R. Simmons, S. Singh, J. M. Snider, A. T. Stentz, W. R. L. Whittaker, and J. Ziglar, "Tartan racing: A multi-modal approach to the darpa urban challenge," Tech. Rep. CMU-RI-TR-, Robotics Institute, Pittsburgh, PA, April 2007. 2.1

[3] J. Wei, J. M. Snider, J. Kim, J. M. Dolan, R. Rajkumar, and B. Litkouhi, "Towards a viable autonomous driving research platform," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 763–770, IEEE, 2013. 2.1

[4] C. R. Baker, D. Ferguson, and J. M. Dolan, "Robust mission execution for autonomous urban driving," in *10th International Conference on Intelligent Autonomous Systems (IAS 2008)*, pp. 155–163, July 2008. 2.1

[5] D. Ferguson, C. R. Baker, M. Likhachev, and J. M. Dolan, "A reasoning frame-

work for autonomous urban driving," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV 2008)*, (Eindhoven, Netherlands), pp. 775–780, June 2008. 2.1

[6] D. Ferguson, T. M. Howard, and M. Likhachev, "Motion planning in urban environments," *Journal of Field Robotics*, vol. 25, no. 11-12, pp. 939–960, 2008. 2.1

[7] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006. 2.1

[8] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010. 2.1

[9] W. Xu, J. Wei, J. M. Dolan, H. Zhao, and H. Zha, "A real-time motion planner with trajectory optimization for autonomous vehicles," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 2061–2067, IEEE, 2012. 2.1

[10] J. Wei, J. M. Dolan, and B. Litkouhi, "Autonomous vehicle social behavior for highway entrance ramp management," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 201–207, IEEE, 2013. 2.1

[11] J. Wei, J. M. Dolan, and B. Litkouhi, "A behavioral planning framework for autonomous driving," in *Intelligent Vehicles Symposium (IV)*, IEEE, 2014. 2.1

[12] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable

bounds on sub-optimality," *Advances in Neural Information Processing Systems (NIPS)*, vol. 16, 2004. 2.2

[13] S. LaValle and J. Kuffner Jr, "Randomized kinodynamic planning," in *Robotics and Automation (ICRA), IEEE International Conference on*, vol. 1, pp. 473–479, 1999. 2.2, 2.2.1

[14] N. Ratliff, M. Zucker, J. A. D. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2009. 2.2

[15] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*, pp. 4569–4574, IEEE, 2011. 2.2

[16] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization.," in *Robotics: science and systems*, vol. 9, pp. 1–10, Citeseer, 2013. 2.2

[17] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958. 2.2.1

[18] L. R. Ford Jr, "Network flow theory," tech. rep., Rand Corp Santa Monica Ca, 1956. 2.2.1

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959. 2.2.1

[20] A. B. Kahn, "Topological sorting of large networks," *Communications of the ACM*, vol. 5, no. 11, pp. 558–562, 1962. 2.2.1

[21] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. 2.2.1

[22] S. Koenig and M. Likhachev, "Dˆ* lite," in *Eighteenth National Conference on Artificial Intelligence (AAAI)*, (Menlo Park, CA, USA), pp. 476–483, American Association for Artificial Intelligence, 2002. 2.2.1, 2.2.2

[23] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain," in *Robotics and Automation (ICRA), IEEE International Conference on*, vol. 1, pp. 968–975, 2002. 2.2.1

[24] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm.," in *ICAPS*, vol. 5, pp. 262–271, 2005. 2.2.1

[25] J. Van Den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 2366–2371, IEEE, 2006. 2.2.1

[26] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996. 2.2.1

[27] J. van den Berg and M. Overmars, "Roadmap-based motion planning in dynamic environments," *Robotics, IEEE Transactions on*, vol. 21, no. 5, pp. 885–897, 2005. 2.2.1

[28] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frenet frame," in *Robotics and Automation (ICRA), IEEE International Conference on*, pp. 987–993, 2010. 2.2.2, 2.1

[29] B. Nagy and A. Kelly, "Trajectory generation for car-like robots using cubic curvature polynomials," *Field and Service Robots*, vol. 11, 2001. 2.2.2

[30] T. M. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *The International Journal of Robotics Research*, vol. 26, no. 2, pp. 141–166, 2007. 2.2.2

[31] A. Kelly and B. Nagy, "Reactive nonholonomic trajectory generation via parametric optimal control," *The International Journal of Robotics Research*, vol. 22, no. 7-8, p. 583, 2003. 2.2.2, 2.1, 3.8, 3.1, 3.2

[32] M. McNaughton, C. Urmson, J. Dolan, and J. Lee, "Motion planning for autonomous driving with a conformal spatiotemporal lattice," in *Robotics and Automation (ICRA), IEEE International Conference on*, vol. 1, pp. 4889–4895, 2011. 2.2.2, 2.1

[33] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo em motion planner," *arXiv preprint arXiv:1807.08048*, 2018. 2.2.2, 2.4, 2.2

[34] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957. 2.2.2

[35] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and

backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990. 2.2.2

[36] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009. 2.2.2

[37] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," in *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*, (Chicago, USA), AAAI, June 2008. 2.2.2

[38] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960. 2.3.1

[39] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference, 2005.*, pp. 300–306, IEEE, 2005. 2.3.1, 3.9

[40] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906–4913, IEEE, 2012. 2.3.1

[41] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, vol. 3, no. 1, pp. 85–95, 1966. 2.3.1

[42] J. P. van den Berg, "Extended lqr: Locally-optimal feedback control for systems

with non-linear dynamics and non-quadratic cost," in *International Symposium on Robotics Research (ISRR)*, 2013. 2.3.1, 2.1, 3.8, 3.1, 3.2

[43] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1168–1175, IEEE, 2014. 2.3.1, 2.1

[44] A. Sideris and L. A. Rodriguez, "A riccati approach for constrained linear quadratic optimal control," *International Journal of Control*, vol. 84, no. 2, pp. 370–380, 2011. 2.3.1

[45] J. Chen, W. Zhan, and M. Tomizuka, "Constrained iterative lqr for on-road autonomous driving motion planning," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–7, IEEE, 2017. 2.3.1

[46] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," in *Fast motions in biomechanics and robotics*, pp. 65–93, Springer, 2006. 2.3.1, 3.9

[47] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998. 2.3.1

[48] H. G. Bock and K.-J. Plitt, "A multiple shooting algorithm for direct solution of optimal control problems," *IFAC Proceedings Volumes*, vol. 17, no. 2, pp. 1603–1608, 1984. 2.3.1, 5.2.1

[49] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of interior-point methods to model predictive control," *Journal of optimization theory and applications*, vol. 99, no. 3, pp. 723–757, 1998. 2.3.1, 3.7.2, 3.7.2

[50] J. Mattingley and S. Boyd, "Cvxgen: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012. 2.3.1, 2.3.2

[51] A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones, "Efficient interior point methods for multistage problems arising in receding horizon control," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 668–674, IEEE, 2012. 2.3.1, 2.3.2, 3.7.2

[52] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017. 2.3.1

[53] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. New York, NY, USA: Cambridge University Press, 2nd ed., 2009. 2.3.1

[54] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*, vol. 2. Nob Hill Publishing Madison, WI, 2017. 2.3.2

[55] J. Nilsson, P. Falcone, M. Ali, and J. Sjöberg, "Receding horizon maneuver generation for automated highway driving," *Control Engineering Practice*, vol. 41, pp. 124–133, 2015. 2.3.2

[56] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015. 2.3.2, 2.1, 3.4

[57] C. Liu, C.-Y. Lin, and M. Tomizuka, "The convex feasible set algorithm for real time optimization in motion planning," *SIAM Journal on Control and Optimization*, vol. 56, no. 4, pp. 2712–2733, 2018. 2.3.2

[58] P. F. Lima, M. Trincavelli, J. Mårtensson, and B. Wahlberg, "Clothoid-based model predictive control for autonomous driving," in *2015 European Control Conference (ECC)*, pp. 2983–2990, IEEE, 2015. 2.3.2

[59] M. G. Plessen, P. F. Lima, J. Mårtensson, A. Bemporad, and B. Wahlberg, "Trajectory planning under vehicle dimension constraints using sequential linear programming," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–6, IEEE, 2017. 2.3.2

[60] F. Altché, P. Polack, and A. de La Fortelle, "High-speed trajectory planning for autonomous vehicles using a simple dynamic model," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–7, IEEE, 2017. 2.3.2

[61] E. Bakker, L. Nyborg, and H. B. Pacejka, "Tyre modelling for use in vehicle dynamics studies," tech. rep., SAE Technical Paper, 1987. 2.3.2

[62] A. Domahidi, "Forces: Fast optimization for real-time control on embedded systems (2012)," *URl: http://forces. ethz. ch*, 2012. 2.3.2

[63] B. Houska, H. J. Ferreau, and M. Diehl, "Acado toolkit-an open-source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011. 2.3.2, 3.8, 3.1, 3.2, 3.3

[64] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *The international journal of robotics research*, vol. 5, no. 3, pp. 72–89, 1986. 2.4, 4.1, 4.2.3

[65] T. Fraichard and C. Laugier, "Path-velocity decomposition revisited and applied to dynamic trajectory planning," in *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pp. 40–45, IEEE, 1993. 2.4

[66] J. Johnson and K. Hauser, "Optimal acceleration-bounded trajectory planning in dynamic environments along a specified path," in *2012 IEEE International Conference on Robotics and Automation*, pp. 2035–2041, IEEE, 2012. 2.4, 4.1, 4.2.3

[67] T. Gu, J. Snider, J. M. Dolan, and J.-w. Lee, "Focused trajectory planning for autonomous on-road driving," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 547–552, IEEE, 2013. 2.4, 2.2

[68] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, 2009. 2.4, 4.1, 4.2.5, 4.3, 4.2

[69] T. Lipp and S. Boyd, "Minimum-time speed optimisation over a fixed path," *International Journal of Control*, vol. 87, no. 6, pp. 1297–1311, 2014. 2.4, 2.2

[70] F. Debrouwere, W. Van Loock, G. Pipeleers, Q. T. Dinh, M. Diehl, J. De Schutter, and J. Swevers, "Time-optimal path following for robots with convex–concave constraints using sequential convex programming," *IEEE Transactions on Robotics*, vol. 29, no. 6, pp. 1485–1495, 2013. 2.4

[71] C. Liu, W. Zhan, and M. Tomizuka, "Speed profile planning in dynamic environments via temporal optimization," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 154–159, IEEE, 2017. 2.4, 2.2, 2.5, 4.1

[72] A. De Luca, G. Oriolo, and C. Samson, "Feedback control of a nonholonomic car-like robot," in *Robot motion planning and control*, pp. 171–253, Springer, 1998. 3.2

[73] P. Rouchon, M. Fliess, J. Lévine, and P. Martin, "Flatness, motion planning and trailer systems," in *IEEE Conference on Decision and Control*, vol. 3, pp. 2700–2700, INSTITUTE OF ELECTRICAL ENGINEERS INC (IEE), 1993. 3.3.1

[74] M. Fliess, J. Lévine, P. Martin, and P. Rouchon, "Flatness and defect of nonlinear systems: introductory theory and examples," *International journal of control*, vol. 61, no. 6, pp. 1327–1361, 1995. 3.3.1

[75] C. H. Reinsch, "Smoothing by spline functions," *Numerische mathematik*, vol. 10, no. 3, pp. 177–183, 1967. 3.5, 3.9

[76] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, 2020. 3.5, 4.2.6, 4.3

[77] A. Nemirovskii and Y. Nesterov, "Interior point polynomial algorithms in convex programming," *Society for Industrial and Applied Mathematics*, 1994. 3.5

[78] M. Andersen, J. Dahl, Z. Liu, L. Vandenberghe, S. Sra, S. Nowozin, and S. Wright, "Interior-point methods for large-scale cone programming," *Optimization for machine learning*, vol. 5583, 2011. 3.5

[79] G. Frison, *Numerical methods for model predictive control*. PhD thesis, University of Padua, 2012. 3.7.2

[80] Y. Wang and S. Boyd, "Fast model predictive control using online optimiza-

tion," *IEEE Transactions on control systems technology*, vol. 18, no. 2, pp. 267–278, 2009. 3.7.2

[81] G. Frison and M. Diehl, "Hpipm: a high-performance quadratic programming framework for model predictive control," 2020. 3.7.2

[82] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006. 3.8, 3.1, 3.2, 3.3, 4.3

[83] M. Althoff, M. Koschi, and S. Manzinger, "Commonroad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intelligent Vehicles Symposium*, pp. 719 – 726, 2017. 3.8.2

[84] Y. Zhang, H. Chen, S. L. Waslander, T. Yang, S. Zhang, G. Xiong, and K. Liu, "Speed planning for autonomous driving via convex optimization," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1089–1094, IEEE, 2018. 4.1

[85] J. E. Bobrow, S. Dubowsky, and J. S. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The international journal of robotics research*, vol. 4, no. 3, pp. 3–17, 1985. 4.1

[86] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," *Robotics: Science and Systems VIII*, pp. 1–8, 2012. 4.1

[87] Q.-C. Pham, "A general, fast, and robust implementation of the time-optimal path parameterization algorithm," *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1533–1540, 2014. 4.1

[88] R. Rajamani, *Vehicle dynamics and control*. Springer Science & Business Media, 2011. 4.2.3

[89] M. Elbanhawi, M. Simic, and R. Jazar, "In the passenger seat: investigating ride comfort measures in autonomous cars," *IEEE Intelligent Transportation Systems Magazine*, vol. 7, no. 3, pp. 4–17, 2015. 4.2.3

[90] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016. 4.2.6, 4.3

[91] B. Chazelle, "Approximation and decomposition of shapes," *Algorithmic and Geometric Aspects of Robotics*, vol. 1, pp. 145–185, 1985. 4.2.8

[92] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006. 4.2.8

[93] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019. 4.3

[94] W. Xu, J. Pan, J. Wei, and J. M. Dolan, "Motion planning under uncertainty for on-road autonomous driving," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2507–2512, IEEE, 2014. 5.2.3