Improved Trajectory Planning for On-Road Self-Driving Vehicles Via Combined Graph Search, Optimization & Topology Analysis

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Department of Electrical & Computer Engineering

Tianyu Gu

Doctoral

Carnegie Mellon University
Pittsburgh, PA

May 2017

Acknowledgements

I must first thank my family, who have been very patient with me and supportive of my academic pursuit.

I would like to thank my advisor, Professor John Dolan, who helped me throughout my Ph.D study. It is in those lengthy discussions that I honed my logical thinking ability and pruned away those obviously dead-end approaches to try for. I sincerely thank my committee members, Professor Howie Choset, Professor Gary Overett and Dr. Jin-Woo Lee for their valuable inputs.

I would like to thank all former colleagues in the GM-CMU ADCRL: Professor Raj Rajkumar, Jarrod Snider, Junqing Wei, Gaurav Bhatia, Junsung Kim, Jongho Lee, Hyunggi Cho, Wenda Xu, Chiyu Dong, Adam Werries and Hyunsung Kim. Their support was critical.

I would like to thank all former colleagues that I worked with at different companies: Bakhtiar Litkouhi/Jin-Woo Lee/Priyantha Mudalige at General Motors, Raj Rajkumar/Jarrod Snider at Ottomatika (now Delphi), Nathan Fairfield/Ioan Alexandru Sucan at Google X (now Waymo), and Tony Stentz/David Bradley/Mike Phillips/Zico Kolter at Uber ATG. My work experiences brought me new perspectives on autonomous driving.

I would also like to thank all those friends that I was fortunate enough to meet and get to know during my Ph.D study in Pittsburgh.

Abstract

Trajectory planning is an important component of autonomous driving. It takes the result of route-level navigation plan and generates the motion-level commands that steer an autonomous passenger vehicle (APV). Prior work on solving this problem uses either a sampling-based or optimization-based trajectory planner, accompanied by some high-level rule generation components. However, these schemes are limited in the following respects:

1. Sampling-based planners yield global resolution-complete optimal trajectories, but suffer from the curse of dimensionality and sampling sub-optimality (i.e., almost never reach a local optimum). Optimization-based planners can find a local optimal trajectory, but suffer from the unawareness of global optimum.

2. Both types of trajectory planner lack explicit tactical or behavior-level reasoning capability. They rely on the high-level behavioral decision-making component to make motion-level decisions that do not necessarily comply with the maneuverability of the planner and the APV.

In this thesis, we adapt existing algorithms and propose new methods in the fields of optimal control (Chapter 3 & 6), graph search (Chapter 4) and topological analysis (Chapter 5) to design an improved trajectory planning system. The core contributions of our work are summarized below:

- A hybrid trajectory planner for on-road autonomous driving that maintains the key advantages of both the sampling-based and the optimization-based planners while reducing their limitations.

- A novel type of edge-augmented graph that allows sampling-based planners to numerically approximate certain trajectory optimization methods.

- A novel backward induction method based on topological analysis to perform configuration-space segmentation over a direct acyclic graph (DAG) as an efficient way to explore the topological structure of a global configuration space.
- A novel maneuver pattern distinction method based on trajectory sampling and topological analysis to distinguish region-based, topology-based and overtaking sequence-based patterns for motion-level tactical reasoning.
- Adaptation of the linear quadratic regulator (LQR) controller for model-feasible trajectory smoothing, and of the iterative-LQR (1^{st}-order differential dynamic programming) for focused execution-feasible trajectory optimization.
- Identification of useful principles and functions for constructing cost terms for trajectory evaluation.

Contents

List of Tables

List of Figures

Terminology

Abbreviation

APV: Autonomous Passenger Vehicle

PRM: Probabilistic Road Map

RRT: Rapidly-exploring Random Tree

DP: Dynamic Programming

DoF: Degree of Freedom

UG: Undirected Graph

DCG: Directed Cyclic Graph

DAG: Directed Acyclic Graph

SOMWF: Single-Objective-Multiple-Weighted-Feature

SSSP: Single Source Shortest Path

N-D: N-dimensional

APV Point: 2-D spatial position in (x, y)

APV Pose: 3-D spatial position and orientation in (x, y, θ)

APV State: High-D vector that describes the exact state of an APV, often includes APV Pose and other information like speed, accelerations, etc.

iff: if and only if.

Nomenclature

Path planning: Planning for a geometric curve.

Trajectory planning: Planning for a time-index control sequence or path (spatial state sequence).

Motion planning: A general name for combined path planning and trajectory planning.

Complete: The trajectory planning is guaranteed to terminate in finite time, returning either failure or a valid solution if one exists.

Resolution-complete: The trajectory planning is guaranteed to terminate in finite time, returning either failure or a valid solution if one exists and the resolution parameter is set fine enough.

Probabilistic-complete: The probability of trajectory planning failure approaches zero as the number of samples approaches infinity if a valid solution exists, for sampling-based approach specifically.

Holonomic vs. Nonholonomic: A holonomic robot has the same number of controllable DoF as the total number of the DoF; nonholonomic robot has fewer controllable DoF than the total number of DoF.

Admissible heuristics: In heuristic-based search algorithms, like A* and its variants, admissible heuristics always return an underestimate of the optimal cost-to-go value.

Reference vs. Trajectory: Both reference and trajectory are used to represent a spatiotemporal trace. Reference is a state sequence indexed by the arc-length of a trace. One may use the speed in the state to implicitly infer time in this sequence (implicit representation). Trajectory is a state sequence directly indexed by time (explicit representation).

Chapter 1

Introduction

1.1 Overview

According to the National Highway Traffic Safety Administration [National Highway Traffic Safety Administration, 2013], approximately two million people were injured and more than thirty thousand people were killed in the United States in 2012 alone. Human error was the major cause of car accidents [Wilson and Stimpson, 2010].

Autonomous passenger vehicles (APV), a.k.a. driverless cars, have brought societal attention to the active research field of vehicle autonomy. Efforts like the 2007 DARPA Urban Challenge [Urmson et al., 2008, Montemerlo et al., 2008] and the Google/Uber self-driving car projects have demonstrated the technological possibility of achieving full autonomy in complex urban environments, and herald a bright future of transportation safety, efficiency and accessibility.

An APV operates in two distinct contexts [Urmson et al., 2008]: the unstructured and the structured. The former represents a large free area (e.g. a parking lot) within which the host vehicle can operate without strong structural constraints, while the latter refers specifically to on-road driving on the highway or in urban environments, where the maneuverability of the host vehicle is confined by lane markings and physical barriers. The latter is more challenging. While the spatial search space is confined,

it has a large temporal space to search for in order to safety navigate around obstacles of full speed-range. It is also more meaningful since most driving is done in the structured environment.

In this thesis, we are primarily concerned with the problems of on-road trajectory planning and control. A hybrid trajectory planning framework and individual planning algorithms are developed and organized to achieve better trajectory quality and more informative high-level decision-making capability.

1.2 Motivation

Broadly speaking, there are two fundamental classes of trajectory planning methods: the optimization-based and the sampling-based. In this section, we give an overview of the advantages and disadvantages/limitations of these approaches, which lead to the core motivation of our approach.

1.2.1 Limitations in Sampling-based Methods

Sampling-based methods can find (or report failure to find) a plan being aware of the entire search space, unlike optimization-based methods. If the search space is a pool of sampled trajectories, a sampling-based planner returns the best trajectory from this pool. If the search space is a graph, a sampling-based planner typically returns the best route traversing this graph. Therefore, if the search space is generated such that it is comprehensive with respect to the continuous space of interest, the solution from such a planner yields a plan that is probably closer to a global optimum.

However, the need to discretize the search space means that one can almost never find a true local minimum. In the meantime, there is a constant trade-off between high search space resolution, performance, and computation on the one hand and low resolution, performance, and computation on the other hand. If the search dimension is large (e.g., for spatiotemporal planning), the so-called *curse of dimensionality* is also likely to make it difficult in to find a good trade-off.

1.2.2 Limitations in Optimization-based Methods

Optimization-based methods are good at finding a locally optimal plan, and a good optimizer is typically designed to return after a few hundred iterations (compared to the thousands or even more trajectory evaluations needed for sampling-based planners), which often implies a small computational overhead. The two main disadvantages of optimization-based planners are that: 1) their run-time is often non-deterministic, and 2) they lack the global awareness of sampling-based planners. Indeed, the solution of most optimization-based trajectory planners depend heavily on a good initialization (seeding) trajectory, and often only returns a local optimum that is close to the initialized trajectory, without having explored a significant portion of the state space.

1.2.3 Topological Unawareness

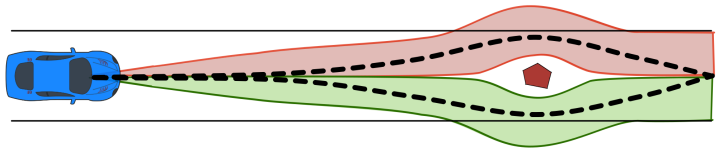
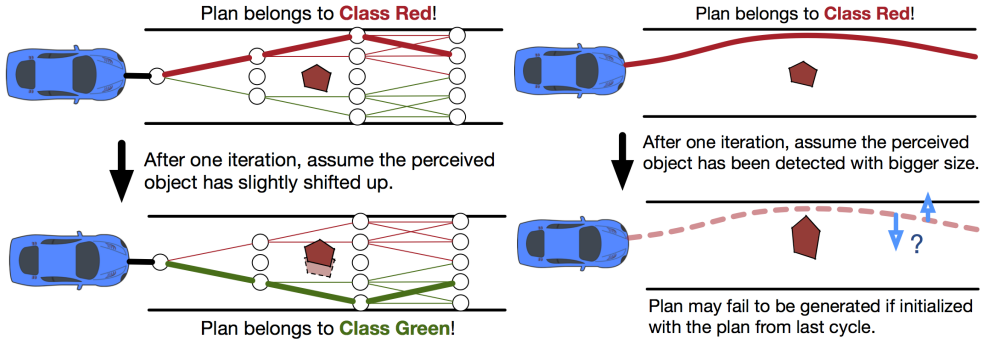


Figure 1.1: Demonstration of topological unawareness with cost function alone.

The existence of obstacles in the environment creates topological structure in the configuration space. Having topological awareness is an important factor in performing high-level tactical reasoning by identifying and reasoning about distinct maneuver patterns. Figure 1.1 illustrates a simple homotopic path planning example, where the vehicle has two topologically distinct maneuvers, swerving to the left or right, and their costs can be close (even equal) to each other.

The lack of topological awareness could lead to further difficulties in both sampling-based and optimization-based methods. For a sampling-based planners, such unawareness could further cause cycle-to-cycle indecisiveness among locally optimal trajectories that belong to drastically different maneuver patterns, as shown in Figure 1.2(a). For an optimization-based planner, such unawareness causes the problem of



(a) Sampling-based (b) Optimization-based

Figure 1.2: Demonstration of topological unawareness leading to problems for both the sampling-based and optimization-based planning techniques.

getting stuck at the wrong local minimum, in an extreme case, can cause planning failure. As shown in Figure 1.2(b), the planner fails to find a plan given the initial collision-inevitable trajectory that belongs to "swerve-left" pattern, even there may exist a feasible solution from the topological alternative "swerve-right" pattern.

1.3 Thesis Statement & Contribution

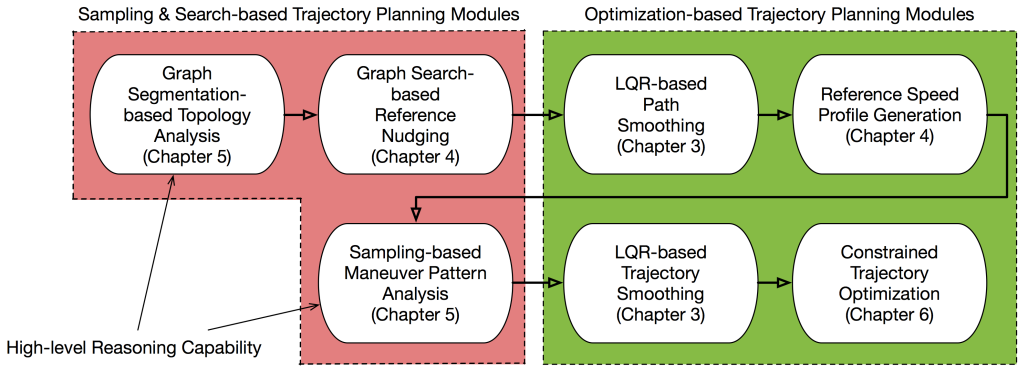


Figure 1.3: The algorithmic flow overview of the proposed planning components.

By taking advantage of a combined sampling-and-search, optimization and topology analysis approach, we can avoid pitfalls of traditional search/optimization-based method and to equip self-driving cars with improved high-level reasoning capabilities for on-road trajectory planning. As shown in Figure 1.3, we adapt existing algorithms and propose new methods in the fields of optimal control (LQR-based Path/Trajectory Smoothing and Constrained Trajectory Optimization in Chapter 3 and 6), graph search (Edge-augmented Graph Search in Chapter 4), and topological analysis (Graph Segmentation/Sampling-based Maneuver Pattern Analysis in Chapter 5). The core contributions of our work are summarized below:

- A hybrid trajectory planner for on-road autonomous driving that maintains the key advantages of both the sampling-based and the optimization-based planners while reducing their limitations.
- A novel type of edge-augmented graph that allows sampling-based planners to numerically approximate certain trajectory optimization methods.
- A novel backward induction method based on topological analysis to perform configuration-space segmentation over a direct acyclic graph (DAG) as an efficient way to explore the topological structure of a global configuration space.
- A novel maneuver pattern distinction method based on trajectory sampling and topological analysis to distinguish region-based, topology-based and overtaking sequence-based patterns for motion-level tactical reasoning.
- Adaptation of the linear quadratic regulator (LQR) controller for model-feasible trajectory smoothing, and of the iterative-LQR (1^{st}-order differential dynamic programming) for focused execution-feasible trajectory optimization.
- Identification of useful principles and functions for constructing cost terms for trajectory evaluation.

Chapter 2

Related Work

This chapter reviews prior work on the planning systems for an APV, and common path/trajectory planning methods for mobile robots. Section 2.1 reviews the hierarchical planning structure that is widely used, and the relationship between levels of planning modules. Section 2.2 reviews three categories of classic trajectory planning approaches that depend on an objective cost function. Section 2.3 reviews homotopy analysis techniques used in trajectory planning. Finally, Section 2.4 summarizes the desired properties of a trajectory planner for autonomous driving vehicles, exposes the shortcomings of prior approaches, and then provides an overview of how the proposed approach addresses them.

2.1 Planning Architecture for Autonomous Driving

From a functional perspective, an APV typically has the following subsystems:

- Localization (self-location on a global/local map)
- Sensing (sensing and constructing a model of the environment)
- Planning/Control (generating and actuating motion safely)

The most common way to organize these different capabilities is through the use of a hierarchical framework. As an example, the framework of "Boss" [Urmson et al.,

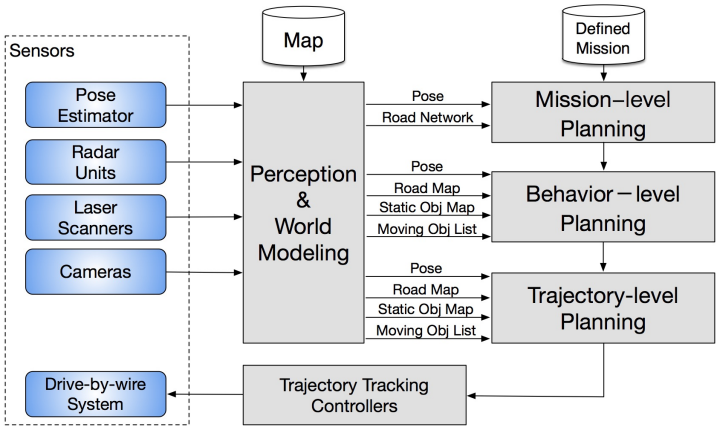


Figure 2.1: The hierarchical overview of CMU's DARPA Ubran Challenge entry "Boss".

2008], winning entry in the 2007 DARPA Urban Challenge race, is shown in Figure 2.1. The focus is in the planning subsystem, which takes on a wide range of navigation and decision making responsibilities. This hierarchy organized these capabilities into three planning levels:

- **Mission-level:** generates a high-level plan composed of a sequence of global checkpoints to be reached to accomplish a navigation task.
- **Behavior-level:** makes mid-level tactical maneuver decisions such as triggering lane-change or staying in-lane, overtaking or distance-keeping, intersection handling or triggering special U-turn/parking-lot maneuvers.
- **Trajectory-level:** generates the low-level trajectory plans, guarantees trajectory feasibility and admissibility by taking into account all constraints from vehicle kinematics/dynamics and surrounding obstacles.

Similar three-level structural planning framework was very popular among all the entries in the 2007 DARPA Urban Challenge that finished the contest course, despite different naming conventions (Table 2.1). The behavior- and trajectory-level planning serve respectively as the key decision maker and motion generator.

Entry	Mission-level	Behavior-level	Motion-level
Boss [Urmson et al., 2008]	Mission Planner	Behavior Planner	Local Planner
Junior [Montemerlo et al., 2008]	Global Path Planner	Behavior Planner	Road Navigator
Odin [Hurdus et al., 2008]	Route Planner	Driving Behavior	Motion Planner
Talos [Leonard et al., 2007]	Mission Planner	Situational Interpreter	Situational Planner
Little Ben [Bohren et al., 2008]	Mission Planner	Local Planner	Path Planner
Skynet [Miller et al., 2008]	Graph Planner	Behavior Planner	Tactical Planner

Table 2.1: Names of the planning modules used by DARPA Urban Challenge Entries.

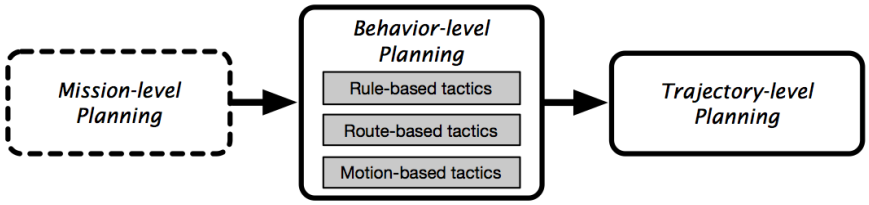


Figure 2.2: The flow of three-level planning and the tactical reasoning responsibilities at the behavior-level.

While the trajectory-level planners generate the executable trajectory to follow, the behavior-level planners are responsible for triggering context-specific motion decisions. A closer look at the behavior-level planner allows one to further distinguish between three tactical layers of planning for on-road driving (Figure 2.2):

- **Rule-based tactics:** stopping & go (precedence handling) at intersection, etc.
- **Route-based tactics:** real-time lane selection to meet global routing/efficiency requirements, etc.
- **Motion-based tactics:** triggering lane-change, to-go at intersection, etc.

Rule-based and route-based tactical planning fall naturally into the behavior-level tasks, and can be efficiently modeled and solved with a Finite State Machine (FSM) as in Urmson [Urmson et al., 2008], a Concurrent Hierarchical State Machine as in

Kammel [Kammel et al., 2008] or more involved Action Selection Mechanisms (ASM) as in Hurdus [Hurdus et al., 2008] and Pirjanian [Pirjanian, 1999].

However, having motion-based tactical reasoning in a behavior-level planner such that it is separated from the actual trajectory planning poses consistency issues: the behavior-level planner may make maneuver decisions that are infeasible from the perspective of the trajectory-level planner, since the actual trajectory has not been generated when reasoning, which as a result does not guarantee motion feasibility. In other words, the motion-based tactical reasoning is dependent on the capability of the trajectory-level planner.

A practical solution to reduce such inconsistency is to design conservative decision-making laws and use expert heuristics to decrease the likelihood of triggering infeasible decisions. However, using conservative rules typically comes at the price of limiting the vehicle's maneuverability, and still provides no consistency guarantee. In addition, it is hard to design behavioral rules for unseen scenarios, e.g., the environment is complicated by multiple moving objects.

2.2 Cost-based Trajectory Planning

An optimal motion planner aims to find a path or trajectory that optimizes a defined objective cost function. We review three classes of methodologies —- reactive (Section 2.2.1), sampling-based (2.2.2) and optimization-based (2.2.3).

2.2.1 Reactive Planning

Reactive planning methods generate an instantaneous control that is responsive to the right-at-the-moment environment.

The potential field (PF) path planning approach by Khatib [Khatib, 1986] Krogh [Krogh and Thorpe, 1986] and Hwang [Hwang and Ahuja, 1992] calculates force fields generated by goal targets and surrounding obstacles: a goal attraction force field and obstacle repulsive force fields. The overlaid effect of these forces determines an im-

mediate control input to the robot. Huang [Huang et al., 2006] and Hamner [Hamner et al., 2006] further extended the original PF method for non-holonomic car-like robots. They incorporated relative headings to the goal and object into the calculation of potential fields, which equivalently generates an angular control law for the robot heading to track a path to the goal. While being physically inspired and easy-to-calculate, the PF approach suffers from possible local trapping due to the existence of local minima in the overall PF function, as discussed by Koren [Koren and Borenstein, 1991].

The navigation function described in [Koren and Borenstein, 1991, Connolly et al., 1990] generates a single minimum globally at the location of the goal configuration with iterative Gauss-Seidel method on discretized cell grid. Lengyel [Lengyel et al., 1990] generates an optimal cost-to-go value for each free configuration sample using a dynamic programming (DP)-based wavefront algorithm. After pre-processing, the cell grid is guaranteed to have a single global minimum at the goal configuration. However, the reason that a navigation function can avoid local optimality is because that the pre-processing step essentially pre-map action for each state.

Overall, it is hard for reactive planning methods to account for the future evolvement (prediction) of the environment in order to yield a deliberative plan that has certain optimality sense overall a long spatial or temporal planning horizon.

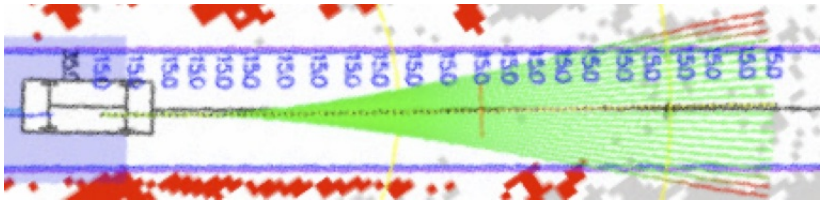
2.2.2 Sampling-based Planning

Sampling-based planning discretizes the continuous configuration space to convert planning into a *generate-and-evaluate* problem or a graph search problem.

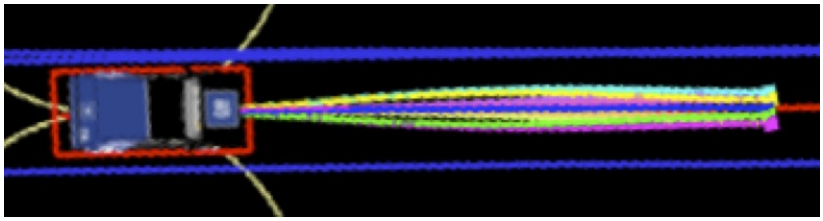
A classical generate-and-evaluate planner is the dynamic window method (DWA), which directly takes into account the constraints imposed by the robot dynamics. In Fox [Fox and Burgard, 1997], the dynamic window is defined by constant translational and rotational velocity commands, resulting in circular trajectories. The optimal trajectory is then selected from the sampled candidates with the minimum cost over the sampled trajectories considering target heading (w.r.t. the goal), clearance to surround-

ing objects, and velocity of the robot.

More involved DMA variants first design certain motion/control primitives that the robot is guaranteed to execute. More precisely, a primitive is a kinematically (dynamically) feasible control sequence/trajectory that connects a pair of start/goal configurations. Piazzi [Piazzi and Bianco, 2000] used quintic polynomials to represent path primitives that connect arbitrary poses in Cartesian space with second-order terminal continuity. Kanayama [Kanayama and Hartman, 1990] proposed to use clothoid curves and cubic spirals, while Nagy [Nagy and Kelly, 2001] developed an efficient routine to generate cubic spirals using shooting-method and gradient descent. Kelly [Kelly and Nagy, 2003] further generalized the problem to generate a spiral curve of arbitrary order in curvature as a model-based boundary satisfaction problem.



(a) Stanley [Thrun et al., 2006]



(b) Boss [Urmson et al., 2008] in 2.3(b)

Figure 2.3: Examples of local sampling patterns.

These motion primitives have been applied to APVs. For example, the winning entries in the DARPA Grand and Urban Challenges, Stanley [Thrun et al., 2006] and Boss [Urmson et al., 2008] made use of the clothoid [Kanayama and Hartman, 1990] and spiral path primitives [Kelly and Nagy, 2003] respectively to generate short-horizon trajectory candidates. The former spawns candidates in a fan-like pattern (Figure 2.3(a)),

whereas the latter samples by laterally shifting the terminal states from the lane centerline (Figure 2.3(b)). Then the optimal trajectory is selected for execution based on a weighted impact of time-optimality, center-closeness and distance to obstacles.

Overall, the *generate-and-evaluate methods* are computationally cheap. However, they are not able to incorporate long-horizon prediction of the environment into the planning process. In fact, these methods can also be viewed as generating a set of candidate trajectories modelled as a single-layer graph. However, no graph search algorithm is necessary, since a simple comparison will find the optimal trajectory.

Graph-based planning methods construct a graph by connecting sampled vertexes (sampled configuration states) with feasible edges (could be a simple linear connection or a more complex primitive, as described above). Planning then becomes searching for an optimal visiting sequence on the graph as the trajectory plan. Depending on how the graph is constructed, the methods can be classified as probabilistic or predefined. Probabilistic methods build the search graph by placing nodes according to a certain statistical distribution (e.g., unbiased randomness means an uniform distribution), whereas the predefined methods perform sampling in a determined pattern. Once a graph is constructed, an appropriate search algorithm to explore and return a solution on this graph is executed.

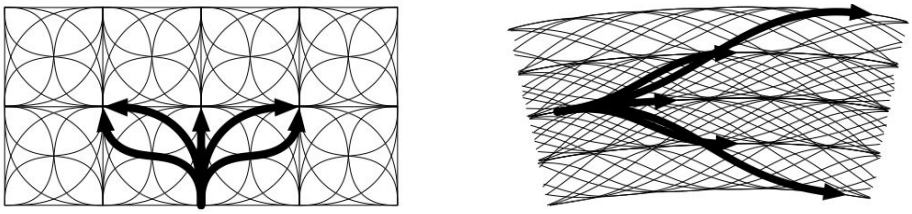
The probabilistic road-map (PRM) method by Kavraki [Kavraki et al., 1996] has been proposed for path planning problems. A learning phase creates a graph, i.e., the road-map, to explore the free configuration by repeatedly growing a tree of randomly sampled configuration points. After connecting the start/goal configurations to the sampled road-map, a query phase then attempts to traverse the road-map by finding a feasible route. Since building the graph consumes the most computational power with query being cheap, PRM routines are suitable for applications that perform multi-query over the same road-map, e.g., in static environments. Work by Le [Le and Plaku, 2014] used PRM to capture the connectivity of the free configuration space, then guided model-based trajectory sampling for efficient maneuver space exploration and search for non-holonomic mobile robots.

The Rapidly-exploring Random Tree (RRT) proposed by LaValle [Lavalle, 1998] can be viewed as a single-query version of PRM. It expands the search tree by iteratively moving toward randomly-sampled configurations. Similar to Le [Le and Plaku, 2014], RRT inherently feeds control to an arbitrarily complex robot model to forward-simulate the vehicle motion as it builds the graph incrementally. However, the original PRM and RRT algorithms are probabilistically complete, but not asymptotically optimal. Theoretically better sampling schemes for both PRM and RRT proposed by Karaman [Karaman and Frazzoli, 2011] result in PRM* and RRT* algorithms to guarantee *asymptotic optimality*, i.e., the global minimum-cost path can always be retrieved given enough sampling.

Variations on RRT implementations are used for APV specifically, commonly biasing the configuration sampling towards the actual goal configurations for better exploration efficiency. Urmson [Urmson and Simmons, 2003] suggested to use path cost as a heuristic to guide the growth of the RRT. Kuwata [Kuwata et al., 2009] specified a goal set according to the high-level navigation destination and road structure to direct RRT growth for MIT's DARPA Urban Challenge team [Leonard et al., 2007]. Jeon [hwan Jeon et al., 2013] designed a RRT*-based motion planner for on-road autonomous driving. The RRT modification in Du [Du et al., 2014] handles narrow passages and cluttered environments by explicitly identifying passable narrow passages in the configuration and performing an obstacle-guided RRT sampling.

Deterministic graph construction methods with predefined-pattern sampling are good alternatives to the probabilistic sampling schemes. Theoretical results from Lavalle [LaValle et al., 2004] even demonstrate the advantages of using deterministic grid graph construction over probabilistic sampling techniques in terms of configuration coverage (dispersion) and practical performance.

Motion primitives, e.g., a spiral [Kelly and Nagy, 2003], have been used in search space construction for a nonholonomic car-like mobile robot. Pivtoraiko [Pivtoraiko et al., 2009] proposed the concept of a state lattice, which embeds a discrete graph composed of kinematically-feasible motion primitives that connect sampled configu-



(a) Open-space lattice (b) On-road lattice

Figure 2.4: State lattices in open-space and on-road environments.

ration states in the continuum (Figure 2.4(a)). Likhachev [Likhachev and Ferguson, 2009] successfully applied this idea to plan long kinematically feasible paths for APV in unstructured environments (e.g., a parking lot) for CMU's DARPA Urban Challenge Entry "Boss" [Urmson et al., 2008].

State lattices in structured environments (Figure 2.4(b)) have been adapted for on-road trajectory planning. Ziegler [Ziegler and Stiller, 2009] created candidate polynomial paths by connecting the robot's pose to a set of sampled poses along the road. Then for each path, a spatiotemporal manifold of linear speed profiles was further sampled to construct a space-time search space. McNaughton [McNaughton, 2011] used a similar sampling pattern with quintic spirals conforming to the geometry of the road. The author also enriched the spatiotemporal search space by denser spatial sampling, allowing internal edge connections and acceleration profile sampling. However, the trajectory candidate pool grows fast due to refined sampling resolution and an extended longitudinal horizon. The computation overhead for trajectory evaluation was compensated with graph pruning techniques based on state bucketing and parallel computation hardware, i.e., GPUs.

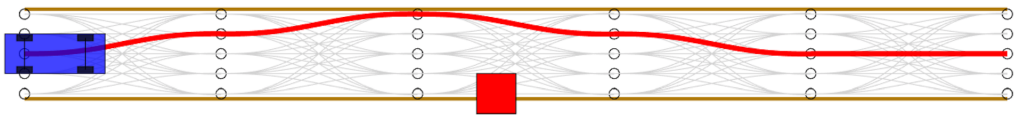


Figure 2.5: Planning sub-optimality caused by fixed lattice sampling resolution.

A lattice created for on-road driving with a long sampling look-ahead horizon is

typically needed for the planner to explicitly reason about the predicted motion of obstacles. A reduction in sampling resolution due to computation resource limitation is needed, which inevitably introduces sampling sub-optimality. In Figure 2.5, a typical spatial lattice is created for on-road driving by laterally sampling poses (black circles) along the road, with the headings chosen to align with the road centerline (for reduced sampling resolution & dimension). Paths (gray curves) are generated to connect sampled poses on neighboring layers. The resulting “optimal" plan (red curve) may cause unnecessary steering along the planned path.

In terms of solving the planning problem, RRT-type algorithms combine graph construction and a graph searching process. But for others, graph search algorithms are applied after the graph is constructed. Algorithms like Depth-first/Breadth-first/Best-first search are easy to use for graph traversal. Iterative deepening [Korf, 1985] extends the Breadth-first search with locally depth-limited depth-first search so that it runs repeatedly with increasing depth limits until the goal is found.

Dijkstra’s algorithm [Dijkstra, 1959] generalizes the Breadth-first search to allow the graph edges to be of non-uniform cost. It finds the minimum-cost path between a source node and every other node (Single-Source, Shortest Path - SSSP). While primarily associated with the greedy process used in Prim’s algorithm, Dijkstra’s algorithm is also closely related to the dynamic programming process, since it successively approximates the optimal cost-to-goal values of all nodes by the reaching method [Sniedovich, 2006]. Thorup [Thorup, 1999, 2003] further accelerated the internal sorting procedure within Dijkstra’s algorithm to achieve linear time complexity, but at the cost of only handling integer cost.

Generalizing Dijkstra’s algorithm, heuristics-based search algorithms, such as A* [Hart et al., 1968], Lifelong A* [Koenig et al., 2004] and D* [Koenig and Likhachev, 2002], have been developed still to find the optimal plan but exploring graph less. A* and its derivatives require an admissible heuristic to guarantee optimality. In other situations, admissibility has to be relaxed for practical considerations. Likhachev [Likhachev et al., 2003] developed a multi-resolution any-time version of A* by relaxing

the admissibility for real-time planning purposes, sacrificing optimality for returning a feasible and error-bounded suboptimal plan in an any-time fashion.

Overall, the sampling-based planning methods, whether probabilistic or predefined, provide a systematic approach for converting the continuous workspace into a discrete workspace modelled by a graph. Established graph traversal/search algorithms can be used to find a probabilistically complete or resolution-optimal planning solution. The lattice-based approaches adapted for both unstructured and structured environments can easily integrate a nonholonomic vehicle's kinematic constraints. On the other hand, the downsides are equally obvious: the curse of dimensionality, the sub-optimality introduced due to discretization, and the potentially intractable computational overhead.

2.2.3 Optimization-based Planning

Optimization-based planning methods iteratively refine a solution until the termination/convergence conditions are met. Optimization is a deep subject, and it is not the main focus of this thesis to exhaustively review optimization techniques. We limit our scope to the optimization techniques used for mobile robots and autonomous driving vehicles. A brief overview of the popular optimization methods used in these domains is provided.

Downhill simplex (Nelder-Mead) method is a geometry-heuristic derivative-free optimization method that can be applied to both linear and nonlinear problems, and is typically used in a high-dimensional problem, where convergence to a local minimum is not required. The *gradient descent method* is a first-order iterative optimization algorithm that finds a local minimum by taking steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. *Newton's method (a.k.a. the Newton-Raphson method)* is a second-order algorithm that further makes use of the hessian information to update both the step size and search direction the accuracy criterion is met. *The Gauss-Newton algorithm* is a modification to

Newton's method that can only be applied to minimize a sum of squares function; it has the advantage of not requiring knowledge of second derivatives. The *Levenberg-Marquardt method* interpolates between the Gauss-Newton algorithm and gradient descent to combine the benefits of both.

Now use the example of solving the linear equation $AX = b$ for X to further introduce several methods, where A is symmetric and positive definite. It is nontrivial to find the inverse of A or use Cholesky decomposition to solve for X directly, especially when A is large and sparse. *Quadratic Programming (QP)* is a special type of convex optimization, whose goal is to optimize a quadratic function $\frac{1}{2}X^TQX + c^TX$. It is not uncommon to reformulate the linear equation solving problem as an unconstrained QP problem $\arg\min F(X) = \frac{1}{2}X^TAX - b^TX$, and iteratively find the X^* that minimizes this function, whose gradient must be zero (the original problem by construction),

$$\nabla F(X^*) = AX^* - b = 0 \tag{2.1}$$

Conceptually, the *gradient method* or *Newton's method* (in just one iteration) can solve the unconstrained QP problem, but finding the inverse of the function's Jacobian is as non-trivial as finding the inverse of A directly or using Cholesky decomposition. The *conjugate gradient descent method* can be used to iteratively solve for this QP problem. It attempts to find several conjugate (A-orthogonal[1]) search directions, which tends to create better search directions for each iteration so that the convergence rate is fast. It becomes more involved if a QP problem has constraints. The equality-constrained QP can be solved by algorithms like *Lagrange Multipliers* or the *null-space (QR factorization) method*. QP with linear inequality constraints $AX \leq b$ can also be solved with algorithms like *Lagrange Multipliers* or *Active Set*.

Oftentimes in practice, we will encounter non-linear or non-convex optimization problems. Physics-inspired methods like simulated annealing [Hwang, 1988] and biological evolution-inspired methods like genetic algorithms [Davis, 1991] are common. They tend to do well in finding global solutions that into account uncertainty (stochas-

[1]Two vectors V and W orthogonal iff $V^TW = 0$; they are conjugate, A-orthogonal, if V^TAW=0.

tic shattering/mutation), but are often very slow. One practical and efficient local method is sequential quadratic programming (SQP). The cost function is locally approximated by a quadratic function through its gradient and hessian, and the dynamics are locally approximated linearly. SQP solves a sequence of QP subproblems, each of which optimizes a quadratic model of the objective subject to a linearization of the constraints.

Optimization techniques used for robot trajectory planning must take into account the robot's dynamic model, which is often given by a differential/finite-difference equation $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ / $\boldsymbol{x}_{i+i} = f(\boldsymbol{x}_i, \boldsymbol{u}_i)$, where $\boldsymbol{x}$ is the state vector of the robots, and $\boldsymbol{u}$ is the control vector. Depending on whether the optimization routine is applied to states or control, there are two classes of trajectory optimization methods – direct and indirect [Von Stryk and Bulirsch, 1992, Betts, 1998]. Direct methods directly alter the state-control-state sequence along the trajectory while enforcing model feasibility between states numerically. Indirect (shooting) methods take the initial state, and only manipulate the controls, and the states along the trajectory must be obtained through forward shooting (a.k.a, integration, evaluation, pass) using the dynamics model.

Many direct methods have been proposed and used for autonomous driving vehicles. Dimitrakakis [Dimitrakakis, 2007] and Thrun [Thrun et al., 2006] used a gradient descent optimizer to minimize the cumulative path jerk by laterally nudging each sampled point within certain lateral or circular bounds. Ziegler [Ziegler et al., 2014] proposed to use SQP to solve a constraint-based non-convex trajectory optimization problem by locally optimizing the state sequence of the trajectory with internal (kinematics) and external (obstacle) constraints. Dolgov [Dolgov et al., 2008] proposed a hybrid trajectory planning approach by first generating a coarse initial path via A*-type search on a graph constructed by primitives obtained with a Reeds-Shepp car model, then locally optimizing with a conjugate gradient descent optimizer.

The elastic-band approaches [Quinlan and Khatib, 1993, Brock and Khatib, 2000, Brandt et al., 2007] are also direct methods. They generate a plan by gradually deforming a coarse path in the configuration space according to artificial forces until

an equilibrium is achieved. Fraichard [Kurniawati and Fraichard, 2007, Fraichard and Delsart, 2009] developed a trajectory deformer (Teddy) to further take the temporal aspect of robot motion into account. The equilibrium can be found with the gradient-descent method or any physics-engine. A recent effort by Roesmann [Roesmann et al., 2012] proposed the so-called Timed Elastic Band (TEB). It further takes the dynamics constraints of the robot into account, as well as the temporal aspects of the motion explicitly by augmenting a time interval between two consecutive configurations.

For indirect(shooting) methods, an extensive survey can be found in [Betts, 1998]. We look at a second-order shooting method —- Differential Dynamic Programming (DDP) [Jacobson, 1968, Jacobson and Mayne, 1970]. DDP iteratively improves a single trajectory by manipulating the control sequence locally. It uses the dynamic programming formulation to optimize within a "tube" around the current trajectory. The dynamics constraints are guaranteed in each shooting process, so the trajectories (state-control sequence) are always feasible. A classical discrete problem setup is: the continuous system dynamics model $\boldsymbol{f}$ is discretized into a system of dynamics $\boldsymbol{f}_d$ with the finite difference dynamics, N is the total time, and the goal is to minimize the cost function in the form of:

$$J = \frac{1}{2}\sum_{i=0}^{N-1}(\boldsymbol{x}_i^T Q \boldsymbol{x}_i + \boldsymbol{u}_i^T R \boldsymbol{u}_i) + \frac{1}{2}\boldsymbol{x}_N^T Q_f \boldsymbol{x}_N \tag{2.2}$$

A simplified version of DDP is the iterative-Linear-Quadratic Regulator (iLQR) [Li and Todorov, 2004b], which uses iterative linearization of the nonlinear dynamics (robot model) around the current trajectory, uses a linear-quadratic methodology to derive Riccati-like equation, and then improves the trajectory. iLQR is more efficient than DDP by dropping the second-order approximation of the model dynamics, yet it still retains good performance for complex dynamics. It can also use line-search to handle nonlinearity. Further improvements were made by Berg [van den Berg, 2016], who exploited the general similarity between optimal estimation and optimal control, and adapted the idea of using an extended Kalman Filter bidirectionally for smoothing [Yu et al., 2004] into iterative-LQR to create the so-called extended-LQR algorithm,

which tends to converge faster.

When there are no control and search-space constraints, the shooting methods (both for DDP and iLQR) are pure unconstrained optimization problems. However, the robot's control input constraints are typically enforced. Tassa [Tassa et al., 2014] proposed a control-limited DDP that is applied to a humanoid robot. Its main difference from traditional DDP is the injection of a local QP procedure in each iteration of local control updating. They further compared the efficiency of this method with naive control clamping and the squashing function, and demonstrated its superiority.

Trajectory optimization methods are local methods, or trajectory smoothing methods, since typically at least one of the optimizer's terms is smoothness-related. These methods are good at refining a coarse trajectory in a local but continuous space. If optimizing on the state space (direct methods), we get a locally optimal state sequence. However, during the optimization process, it suffers from "over-parametrization". Sometimes enforce control limits and model feasibility by directing nudging the states is optimizer-unfriendly, especially when the dynamics model is complicated. On the other hand, if optimizing on the control space (indirect methods), both control limits and model feasibility can be guaranteed through an add-on local optimization procedure inside the main loop of the iterative shooting method.

2.3 Motion Planning with Topology Awareness

The inability to directly reason about environment topology can cause significant problems in a pure SOMWF planning formulation, as discussed in Figure 1.1. Topological analysis is a technique used to extract the topological structure of an environment. It observes that topologically distinct trajectory classes arise in the presence of obstacles. By being aware of topological information, the planner can incorporate a certain degree of high-level environment understanding, and thereby reduce the exploration space, or make more knowledgeable and decisive decisions.

Topological information can be extracted by configuration space segmentation meth-

ods like triangulation [Demyen and Buro, 2006], cell-decomposition [Lingelbach, 2004] and visibility-graph [Lozano-Pérez and Wesley, 1979], and probabilistic road-map [Kavraki et al., 1996]. The segmented space can be represented by a graph. Then topology-sensitive graph traversal and listing algorithms to explore all homotopic classes have been used to obtain a topology of the graph, such as the one proposed by Schmitzberger [Schmitzberger et al., 2002]. Recent efforts such as those of Kuderer [Kuderer et al., 2014] and Park [Park et al., 2015, Park et al., 2016] proposed to perform topological analysis for mobile robots with Voronoi diagrams and cell decomposition, respectively. However, these methods are limited to 2-D paths. Moreover, the need to generate the complete topology graph causes exponential growth in its size along with the number of obstacles. Finally, it is unclear how to apply these methods to explicitly reason about the predicted motion of moving obstacles in the spatiotemporal domain.

Other methods directly create a topology graph based on objects that exist in the configuration space. Jenkins [Jenkins, 1991] proposed a method that directly turns the 2D workspace into a topological graph by constructing the reference frame emanating from each individual object. The beam-graph method by Narayanan [Narayanan et al., 2013] encodes the topological information by virtual sensor beams emanating from the obstacles.

These methods are also limited to solving low-dimensional path planning problems, do not explicitly consider the temporal evolvement, and do not reason about the spatiotemporal domain needed for trajectory planning in order to take into account the moving objects encountered in autonomous driving.

Bhattacharya [Bhattacharya et al., 2010, Bhattacharya, 2012] proposed to use the "L-value" inspired by complex analysis [Rudin, 1987] and "homology-signature" inspired by electromagnetic theory [Stratton et al., 2007], and combine them with traditional heuristic search to efficiently determine the homotopic class for 2-D and 3-D path planning. However, this method is not directly applicable to motion planning problems with a moving planning horizon (e.g., on-road motion planning), and does not directly account for non-holonomic vehicle dynamics constraints. Recently, Rosmann

[Rösmann et al., 2015] applied the "L-value" with elastic band to perform homotopy-aware path planning for small mobile cars. But not much has been seen on using the "H-signature" to perform topology-aware planning in the spatiotemporal configuration space.

Other traditional methods have been combined with topology analysis tools. For example, Hernandez [Hernandez et al., 2015] systematically proposed a set of topology-aware transitional algorithms, such as H-A*, H-RRT and H-Bug, where "H" stands for homotopy. Kim [Kim et al., 2012] formulated an optimization-based trajectory generation problem by solving quadratic programming for each iteration to obtain the locally optimal trajectory in the specified homotopy class.

2.4 Summary

To motivate new planning structure and algorithms for on-road autonomous driving, we started with an architectural overview of the planning system for an APV, and exposed the typical problem with a hierarchical planning system caused by segregation between behavioral and motion planning modules. This leads to the first goal of this thesis:

- **Goal 1**: release some responsibility from the behavior planner to allow the motion planner to assume environment understanding and higher-level reasoning capability, which makes the reasoning process explicitly take into account the feasibility and admissibility of the trajectory plan.

We then reviewed traditional cost-based motion planning techniques and the typical ways to devise feature cost functions. There are three distinct classes of solvers: reactive, sampling-based and optimization-based. Reactive methods are computationally efficient, but too short-sighted to achieve global/local optimality. Sampling-based methods provide a systematic way to explore the search space, and can alleviate local optimality issues by finding a resolution-complete global optimum. However, the

curse of dimensionality can make them very computationally expensive and cause them to suffer from resolution suboptimality. Optimization-based methods, if properly designed and initialized, can return a local optimal plan through a few iterations, therefore computationally desirable. But the lack of global awareness may cause the planner to converge to the wrong local minimum. These facts lead to the second goal of this thesis:

- **Goal 2**: use a combination of different planning methods to preserve the best of each type for autonomous driving motion planning: control-based for initialization, search-based for avoiding local optimality, and optimization-based for local refinement and smoothing.

Finally, we looked at a different category of motion planning approach: the topology-aware planning approach. Topology information is sometimes very useful to allow the robotic system to better understand the environment and make smarter decisions. However, the traditional planning approaches are dependent on feature cost functions, in which it is very difficult to encode topological information. This leads to the third and final goal of this thesis:

- **Goal 3**: create an efficient topological awareness technique and combine it with traditional sampling-based and optimization-based methods. Use topological information as one of the main indicators to distinguish distinct maneuver patterns, and support higher-level reasoning.

Chapter 3

Vehicle Model, Representation & Control

In this chapter, we first review two important vehicle dynamics models and methods to represent the shape of polygonal objects. Then we review the popular linear quadratic regulator (LQR) controller and an efficient formulation to perform trajectory tracking control for smoothing purposes.

3.1 Vehicle Model

3.1.1 Kinematic Bicycle Model

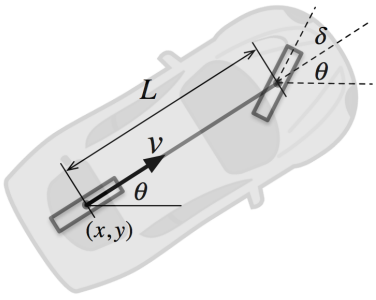


Figure 3.1: Kinematic bicycle model.

Under normal operation, it is typically a valid assumption that the vehicle moves without skidding. So it is common to use the kinematic bicycle model (a.k.a the half-car model) to describe the motion of an APV (Figure 3.1). Restricting motion to the plane, the constraints of no lateral motion of both front and rear wheels can be represented by:

$$\begin{aligned} \dot{x}_f sin(\theta + \delta) - \dot{y}_f cos(\theta + \delta) &= 0 \\ \dot{x} sin(\theta) - \dot{y} cos(\theta) &= 0 \\ x + L cos(\theta) &= x_f \\ y + L sin(\theta) &= y_f \end{aligned} \tag{3.1}$$

Eliminating (x_f, y_f), we get

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

Replacing $\dot{x}, \dot{y}$ with $(v \cdot cos(\theta), v \cdot sin(\theta))$, we get

$$\dot{\theta} = \frac{tan(\delta)}{L} \cdot v \tag{3.3}$$

Overall, the kinematics can be written as:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\delta} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & cos(\theta) \\ 0 & 0 & 0 & 0 & sin(\theta) \\ 0 & 0 & 0 & 0 & \frac{tan(\delta)}{L} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \\ \delta \\ v \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \gamma \\ a \end{bmatrix} \tag{3.4}$$

where

- (x, y, θ): APV pose at rear differential (state).
- δ: steering angle of the road wheel (state).
- v: longitudinal speed (state).
- γ: rate of the steering angle (control).

- a: longitudinal acceleration (control).
- L: the length of the wheelbase (parameter).

3.1.2 Dynamic Bicycle Model

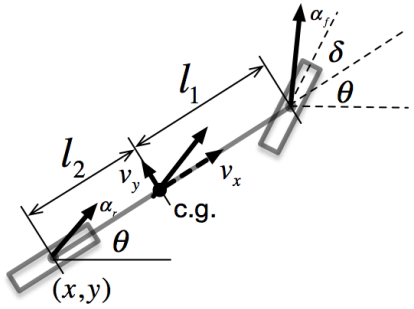


Figure 3.2: Dynamic bicycle model.

The kinematic bicycle model models no lateral dynamics, since it assumes no slipping is possible. While this assumption is valid for low-speed navigation on a dry asphalt road, it is not valid at high speed or on slippery roads. So a more complex model [Bevly et al., 2006] with lateral dynamics was proposed for refined motion dynamics (Figure 3.2):

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\delta} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{C_{\alpha f}}{m} & 0 & \frac{-C_{\alpha f}-C_{\alpha r}}{m v_x} & -v_x + \frac{C_{\alpha r} l_2 - C_{\alpha f} l_1}{m v_x} \\ 0 & 0 & 0 & \frac{C_{\alpha f} l_1}{I_z} & 0 & \frac{C_{\alpha f} l_2 - C_{\alpha f} l_1}{I_x v_x} & \frac{-C_{\alpha f} l_1^2 - C_{\alpha r} l_2^2}{I_z u_x} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ \theta \\ \delta \\ v_x \\ v_y \\ r \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \gamma \\ a \end{bmatrix} \tag{3.5}$$

where

- (x, y, θ): APV pose at rear differential (state).

- v_x, v_y: longitudinal/lateral velocity at center of gravity (state).
- r: yaw rate (state).
- γ: rate of the steering angle (control).
- a: longitudinal acceleration along the APV heading (control).
- $C_{\alpha f}, C_{\alpha r}$: front and rear tire cornering stiffness (parameter).
- l_1, l_2: distance from the center of gravity to front/rear axles (parameter).
- m: vehicle mass (parameter).
- I_z: yaw moment of inertia of the APV (parameter).

More involved vehicle models and commercial packages are also available, but for planning purposes, the two models given in this section are sufficient in most cases. Both models can be used for subsequent tracking control and path smoothing or in the shooting process for trajectory optimization in Chapter 6.

3.1.3 Numerical Integration of Model Dynamics

Sometimes it is important to get a discrete version of the vehicle dynamics by integrating the equation over a short period of time. To get more accurate forward prediction over a short period of time, we can also obtain the discrete-time model by performing numerical integration with various methods, e.g., the Runge–Kutta 4-th order method:

$$\boldsymbol{x_{i+1}} = \boldsymbol{f_d}(\boldsymbol{x_i}, \boldsymbol{u_i}) = \boldsymbol{x_i} + \frac{\boldsymbol{\delta t}}{\boldsymbol{6}}(\boldsymbol{k_1} + \boldsymbol{2k_2} + \boldsymbol{2k_3} + \boldsymbol{k_4}) \tag{3.6}$$

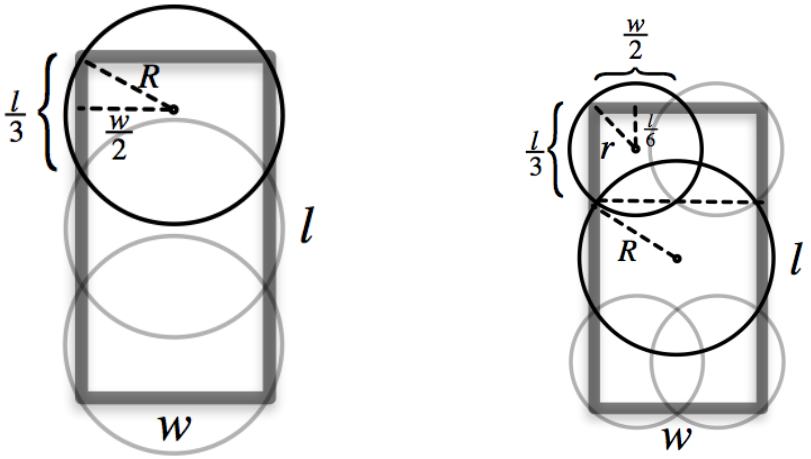
where

$$\begin{aligned} k_1 &= f(x_i, u_i) \\ k_2 &= f(x_i + \frac{\delta t}{2} k_1, u_i) \\ k_3 &= f(x_i + \frac{\delta t}{2} k_2, u_i) \\ k_4 &= f(x_i + \delta t k_3, u_i) \end{aligned}$$

3.2 Representation of APV and Objects

Vehicles and other objects are commonly modeled as 3-DOF planar objects that can translate and rotate in a 2-D environment. The representation matters a lot in terms of efficiently retrieving obstacle-related distance information, which is commonly used in the calculation of a feature cost (See Chapter 6).

Depending on their nature, stationary objects are stored in an occupancy grid (OG), since they oftentimes do not have a concrete and separable shape. Operations like the distance transform can be used to find the nearest distance from a point to any occupied cell. Applying shape convolution before using the distance transform can be used to pre-process the OG to efficiently find the nearest distance from the shape in question to any occupied cell. The downside is that the representation accuracy is subject to the resolution.



(a) Three-disk representation

(b) Five-disk representation

Figure 3.3: Representation of rectangular-shaped objects with circular disks.

In practice, a lot of real world objects can be reasonably approximated by simple geometric shapes like circles and rectangles. The distance computation between two circles is trivial, and therefore ideal for efficiency concerns. When suitable, replacing rectangular objects with multiple circles is also very useful in terms of improving

computational efficiency. For example, Figure 3.3 demonstrates two such possibilities. Figure 3.3(a) demonstrates replacing a rectangle with three circular disks, in which the radius R is given by:

$$R = \sqrt{(\frac{w}{2})^2 + (\frac{l}{6})^2} \tag{3.7}$$

And Figure 3.3(b) demonstrates replacing a rectangle with five circular disks, in which the large disk radius R is calculated as above, and the small disk radius r is given by:

$$r = \sqrt{(\frac{w}{4})^2 + (\frac{l}{6})^2} \tag{3.8}$$

3.3 Trajectory Tracking Control

Suppose we are given a coarse trajectory plan that is not guaranteed to be dynamically feasible, or even non-smooth, and the objective is to find a sequence of control/states that attempts to follow this course plan. Obviously, it is numerically difficult to directly convert a coarse plan into a control sequence due to non-smoothness and lack of model-based evaluation.

A tempting solution is to use spline interpolation to try to smooth out the coarse trajectory, then use the locally parametric representation of the spline and some model simplification to analytically convert into a control sequence. However, parametric splines, if chosen naively, may introduce undesirable jerkiness due to over-fitting, and model simplification may be unrealistic, which causes dynamically infeasible control sequences.

Therefore, we make use of a closed-loop control scheme to initialize the control sequence by attempting to track the non-smooth coarse trajectory plan. Trajectory tracking is typically decoupled into lateral (path tracking) control and longitudinal (speed tracking) control. Either can be tracked with different kinds of controllers, such as geometric pure pursuit, PID-tracker or linear quadratic regulator (LQR). LQR is preferred for its optimality and tunability.

The system dynamics are represented by the following differential equation:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{3.9}$$

We first perform the first-order Taylor expansion around a point $(\boldsymbol{x}^*, \boldsymbol{u}^*)$,

$$\dot{\boldsymbol{x}} \approx f(\boldsymbol{x}^*, \boldsymbol{u}^*) + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(\boldsymbol{x} - \boldsymbol{x}^*) + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}(\boldsymbol{u} - \boldsymbol{u}^*) \tag{3.10}$$

Moving terms around, we get the locally linearized system:

$$\dot{\bar{\boldsymbol{x}}} = A\bar{\boldsymbol{x}} + B\bar{\boldsymbol{u}} \tag{3.11}$$

where

$$\bar{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x}^*$$
$$A = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(\boldsymbol{x}^*, \boldsymbol{u}^*)$$
$$B = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}(\boldsymbol{x}^*, \boldsymbol{u}^*)$$

Supposing the general (possibly non-quadratic) cost function of a given state/control pair is given by $g(\boldsymbol{x}, \boldsymbol{u})$, we can quadratize around a point $(\boldsymbol{x}^*, \boldsymbol{u}^*)$:

$$g(\boldsymbol{x}, \boldsymbol{u}) \approx g(\boldsymbol{x}^*, \boldsymbol{u}^*) + \frac{\partial g}{\partial \boldsymbol{x}}\bar{\boldsymbol{x}} + \frac{\partial g}{\partial \boldsymbol{u}}\bar{\boldsymbol{u}} + \frac{1}{2}\bar{\boldsymbol{x}}^T\frac{\partial^2 g}{\partial \boldsymbol{x}^2}\bar{\boldsymbol{x}} + \frac{1}{2}\bar{\boldsymbol{x}}^T\frac{\partial^2 g}{\partial \boldsymbol{x}\partial \boldsymbol{u}}\bar{\boldsymbol{u}} + \frac{1}{2}\bar{\boldsymbol{u}}^T\frac{\partial^2 g}{\partial \boldsymbol{u}^2}\bar{\boldsymbol{u}} \tag{3.12}$$

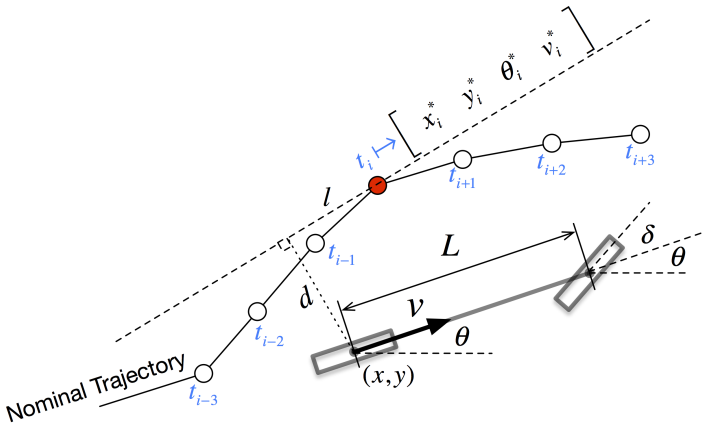


Figure 3.4: Nominal trajectory sequence and the representation of longitudinal/lateral errors with respect to the corresponding nominal trajectory point.

The successful application of the LQR controller depends on two factors. First, the system controlled must demonstrate or can be closely approximated by a linear

system. Then, there must be a way to define the quality of the control task or to approximate it by a quadratic cost function. The crux then becomes to formulate the trajectory tracking task in a way that can be linearized and quadratically measured.

In the application of trajectory tracking control, system linearization is achieved by converting the APV's motion dynamics into error dynamics after performing a coordinate transformation into a nominal coordinate system[1] similar to that described in [Snider, 2009], but extended to spatiotemporal points[2] (Figure 3.4). We assume the vehicle model is a kinematic model that takes in steering angle rate and longitudinal acceleration. At any given time, the nominal point $\boldsymbol{p}^*$ is given by $[x^*, y^*, \theta^*, v^*]$, the lateral offset d, and the longitudinal offset l with respect to $\boldsymbol{p}^*$.

The overall control task is further decoupled into lateral tracking and longitudinal tracking. For lateral control, assume the state vector is

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} d \\ \theta^* - \theta \\ \delta \end{bmatrix} \tag{3.13}$$

The lateral offset d is:

$$\begin{aligned} d &= sin\theta^* \cdot x - cos\theta^* \cdot y + (x^* + cos\theta^*) \cdot x^* - (y^* + sin\theta^*) \cdot x^* \\ &= sin\theta^*(x - x^*) - cos\theta^*(y - y^*) \end{aligned} \tag{3.14}$$

Taking the derivative of each dimension of X, and using the kinematic model of the vehicle described in section 3.1 by replacing $\dot{x}$ and $\dot{y}$ with $v \cdot cos\theta$ and $v \cdot sin\theta$,

$$\begin{aligned} \dot{x}_1 &= \dot{d} = v \cdot sin\theta^* \cdot cos\theta - v \cdot cos\theta^* \cdot sin\theta \\ &= v \cdot sin(\theta^* - \theta) = v \cdot sin(x_2) \\ \dot{x}_2 &= -\dot{\theta} = -v \cdot \frac{tan\delta}{L} = -\frac{v}{L} \cdot tan(x_3) \\ \dot{x}_3 &= \dot{\delta} \end{aligned} \tag{3.15}$$

[1] Calculates the longitudinal and lateral distances with respect to waypoints on a nominal trajectory

[2] A 3-D time-sequenced set of spatial waypoints on the coarse trajectory is not necessarily feasible or even continuous

Under the assumption that the heading error $x_2 = \theta^* - \theta$ is small, $sin(x_2)$ can be approximated by x_2. Also assuming small steering angle $x_3 = \delta$, $tan(x_3)$ can also be approximated by x_3. Overall, the equations above can be generalized and approximated with the following linear equation:

$$\begin{aligned} \dot{\boldsymbol{x}} &= A\boldsymbol{x} + Bu \\ \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} &= \begin{bmatrix} 0 & v & 0 \\ 0 & 0 & -\frac{v}{L} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \dot{\delta} \end{aligned} \tag{3.16}$$

For longitudinal control, assume the state space is

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} l \\ v^* - v \end{bmatrix} \tag{3.17}$$

The longitudinal offset l is:

$$\begin{aligned} l &= -cos\theta^* \cdot x - sin\theta^* \cdot y + (x^* + sin\theta^*) \cdot y^* - (y^* - cos\theta^*) \cdot x^* \\ &= -cos\theta^*(x - x^*) - sin\theta^*(y - y^*) \end{aligned} \tag{3.18}$$

Taking the derivative of X, and using the kinematic model of the vehicle described in section 3.1 above,

$$\begin{aligned} \dot{x}_1 = \dot{l} &= -v \cdot cos\theta^* \cdot cos\theta - v \cdot sin\theta^* \cdot sin\theta \\ &= v^* - v \cdot cos(\theta^* - \theta) - v^* \\ \dot{x}_2 &= -\dot{v} = -a \end{aligned} \tag{3.19}$$

Note that under the assumption that the heading error $\theta^* - \theta$ is small, $cos(\theta^* - \theta)$ can be approximated with 1. Overall, the equations above can be generalized and approximated with the following linear equation while ignoring the constant term:

$$\begin{aligned} \dot{\boldsymbol{x}} &= A\boldsymbol{x} + Bu \\ \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} a \end{aligned} \tag{3.20}$$

The lateral and longitudinal linear system $\dot{\boldsymbol{x}} = A\boldsymbol{x} + Bu$ in the nominal coordinate system has been formalized by Equations 3.16 and 3.20. While the original formulations are not strictly linear, they can be linearly approximated nicely given the reasonable small angle assumption, and dropping certain nonlinear terms.

Another requirement for the successful application of LQR is a quadratic cost function. Looking at the state vectors, they include the longitudinal and lateral tracking error terms l and d in their states. We can naturally use a quadratic of X to measure the performance of the tracker, and a quadratic of U to trade off with the control efforts:

$$J = \int_0^\infty (\boldsymbol{x}^T Q \boldsymbol{x} + u^T R u) \cdot dt \tag{3.21}$$

Then the classical LQR control law states that an optimal control at each time-stamp can be achieved (assuming infinite control horizon) with the following control:

$$u = -K \cdot \boldsymbol{x} \tag{3.22}$$

where $K = R^{-1}B^T P$ and P is found by solving the Riccati equation

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{3.23}$$

The initial trajectory is given as a time sequence of simple states in the form of $[x^*, y^*, \theta^*, v^*]$. The trajectory tracker developed above converts an initially infeasible trajectory into a dynamically feasible control/state sequence without particular requirements on the initial trajectory. For example, the initial trajectory may be piecewise linear, or even a single stationary point. Another advantage of the spatiotemporal nominal trajectory tracking formulation is its computational efficiency. This approach does not involve the most computationally expensive operation of most spatial path tracking algorithms, i.e., find-nearest-point on the spatial nominal path (whose complexity is of $\mathcal{O}(N)$, where N is the number of points in the nominal path).

3.4 Controller-based Smoothing

Given a coarse nominal path or trajectory and a selected vehicle model, we let the LQR controller attempt to track from a starting APV state (Figure 3.5). Upon finishing, the

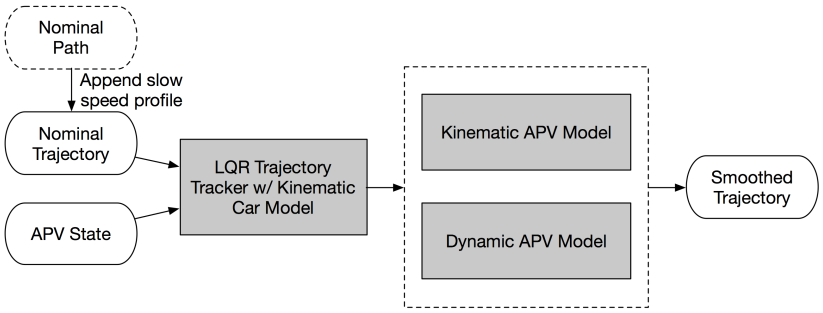


Figure 3.5: Path & trajectory smoothing with LQR-based trajectory tracker.

controller generates a control/state sequence that is guaranteed to be model-feasible[3], which achieves the purpose of smoothing. Depending on the actual model used in forward shooting, the final trajectory can be regarded as kinematically or dynamically feasible. In the meantime, there are typically two types of smoothing tasks: path smoothing and trajectory smoothing.

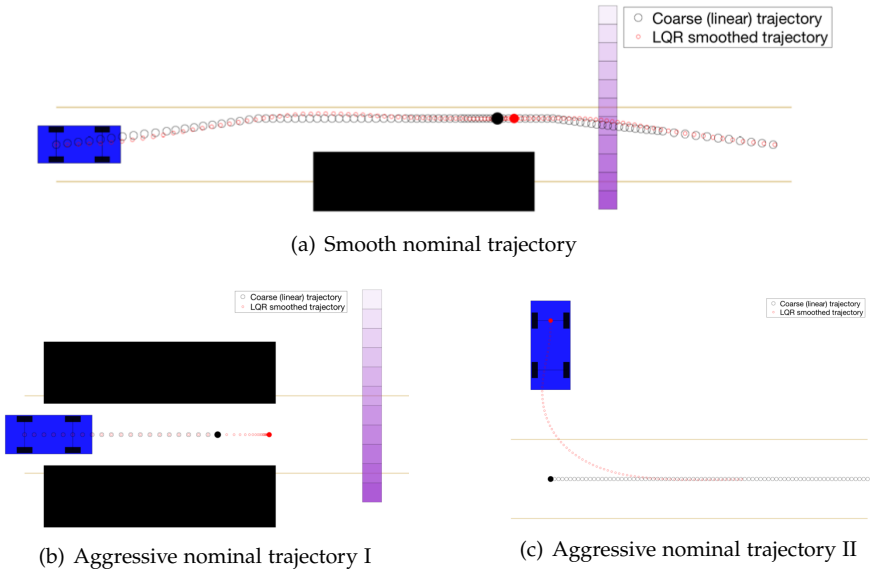
- Path smoothing: convert piecewise linear spatial trace into a smooth path in order to retrieve smooth curvature information (Chapters 4 & 5).
- Trajectory smoothing: convert piecewise-linear spatiotemporal trace into a smooth trajectory to generate a feasible candidate sampling pool, and a good seeding initialization for optimization (Chapter 5 & 6).

We apply the same LQR tracker for both path/trajectory smoothing. Note that for path smoothing, we convert path to a trajectory by appending a constant slow speed (e.g., 1 m/s) to the spatial trace. Regardless of which model is actually used, the final behavior of the car will exhibit the kinematic vehicle model behavior, and the final result will be kinematically feasible. Both the nominal and smoothed trajectories will be represented spaced at every 0.1 second in the following discussion, and visualized in Figure 3.6.

While the (coarse) nominal trajectory is often collision-free, the smoothed trajectory may void this guarantee in exchange for smoothness and model feasibility. However,

[3]Model feasibility is different from executation feasbility, see Chapter 6 for more details

it will not pose a serious problem in the way we use this tracking smoother, since it is used to generate the smoothed seeding trajectory to the final trajectory optimization procedure in Chapter 6, which further refines this smoothed trajectory with collision-free guarantees.



(a) Smooth nominal trajectory

(b) Aggressive nominal trajectory I

(c) Aggressive nominal trajectory II

Figure 3.6: LQR smoothing examples for smooth and aggressive nominal trajectories.

Depending on the quality of the nominal trajectory, the smoothed trajectory may or may not have a large maximum tracking error, which is the maximum point-wise Euclidean distance between the nominal and smoothed trajectories. The definition for "large", however, is very empirical. In the following discussion, we will use 1 meter as an arbitrary threshold. If the maximum tracking error between the correspondingly timed points on the nominal and the smoothed trajectory is less than 1.0, the nominal trajectory will be regarded as "mild", otherwise, it will be regarded as "aggressive".

Figure 3.6(a) demonstrates a mild situation where the piecewise-linear nominal trajectory (black dots) captures the intended behavior: the APV nudges left to avoid a parked bus (black rectangle), slows down for a crossing pedestrian (purple rectangle), and then speed up to resume normal cruising. The smoothed trajectory (red dots) are

close to the nominal trajectory with a maximum tracking error 0.89 meter (distance between the solid black and red dots).

While the intention is to always smooth a mild nominal trajectory, we also want to test how robust the smoothing procedure is with aggressive nominal trajectories. Remember that we made three important assumptions in deriving the LQR controllers: the kinematic model simplification, small angle approximations and constant term ignorance to perform linearization. It is important to justify, at least empirically, these assumptions are valid even in situations not ideal.

Figure 3.6(b) demonstrates an aggressive situation, where the APV is cruising through a narrow corridor created by two parked buses (black rectangles). We impose an abrupt stop on the nominal trajectory (black dots), such that the nominal speed at each point is 5 m/s except for the last point (rightmost solid black dot), to make it difficult for the tracking smoother. It can be seen that the smoothed trajectory (red dots) brought the APV to a full stop with an over-shooting (maximum tracking error) of 2.67 meters (distance between the solid black and red dots). While this tracking error is above the 1 meter threshold, this is still satisfactory given the aggressiveness in speed of the nominal trajectory.

Another extremely difficult tracking (smoothing) task is shown in Figure 3.6(c), where the nominal trajectory (black dots) deviates greatly (8 meters) from where the APV's initial position (red solid dot) and orientation (by 90 degree). In this extreme condition, the tracking controller starts with a huge tracking error, many other tracking algorithm (e.g., pure pursuit, PID controller) may fail to converge. On the other hand, the LQR controller still demonstrates stable behavior by generating a smoothed trajectory (red dots) that brings the APV back to the nominal trajectory. This empirically demonstrates the robustness of using the LQR tracker for trajectory smoothing.

Chapter 4

Edge-augmented Search-based Path Planning

Graphs are commonly used in sampling-based planning methods (Section 2.2.2). Sometimes, graphs are constructed with motion primitives as in the state lattice approaches, e.g., Pivtoraiko et al. [2009], McNaughton [2011], and the result of these planners can be directly used for execution or further smoothed. In this chapter, we construct a special type of edge-augmented graph and search for an optimal plan in a pure path planning setting. There are a few features of this approach:

- For on-road planning, we exploit the fact that the car only drives forward in a corridor-like environment.
- A generic formulation is used to account for road geometry and model simple nudging maneuvers by mimicking the continuous path optimization methods.
- We search over the full sampled configuration space to alleviate the problem of local optimality in optimization methods.

The outcome is an efficient graph search algorithm that numerically approximates continuous optimization-based path smoothing and nudging with resolution-complete global optimality achieved.

4.1 Continuous Path Smoothing & Nudging

A corridor-like environment can be modeled by a densely sampled waypoint sequence representing the center of the corridor, and lateral boundary associated with each point. Intuitively, path planning is achieved by moving the centerline points laterally, subject to the boundary constraints of the corridor.

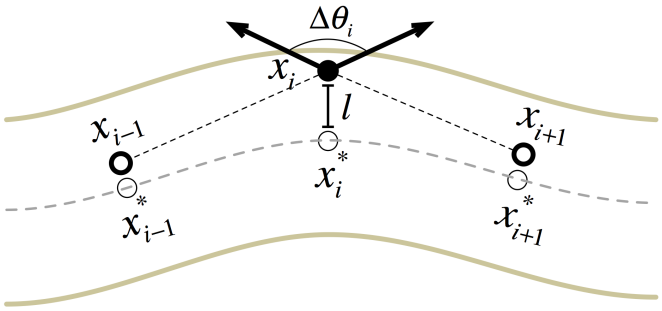


Figure 4.1: The definition of lateral offset and smoothness terms with discrete point samples.

For path smoothing, we typically want to nudge the waypoints so that the overall smoothness of the path is improved (Figure 4.1). Two aspects interactively shape the final path: the closeness to the centerline and a metric for overall smoothness. For each waypoint x_i^* along the centerline, we want to find a new position represented by x_i such that the following cost is minimized:

$$\underset{\{x_i\}}{\operatorname{argmin}} \sum_i \omega_{cl} \cdot \|\boldsymbol{x}_i - \boldsymbol{x}_i^*\|^2 + \omega_{sm} \cdot \frac{(\boldsymbol{x}_i - \boldsymbol{x}_{i-1})(\boldsymbol{x}_i - \boldsymbol{x}_{i+1})}{\|\boldsymbol{x}_i - \boldsymbol{x}_{i-1}\|\|\boldsymbol{x}_i - \boldsymbol{x}_{i+1}\|} \tag{4.1}$$

where the first term is the quadratic distance between the new waypoint and its projection onto the centerline, and the second term is a smoothness measurement that captures the angular change $\Delta\theta_i$ at each waypoint x_i because:

$$\Delta\theta_i = cos^{-1}[\frac{(\boldsymbol{x}_i - \boldsymbol{x}_{i-1})(\boldsymbol{x}_i - \boldsymbol{x}_{i+1})}{\|\boldsymbol{x}_i - \boldsymbol{x}_{i-1}\|\|\boldsymbol{x}_i - \boldsymbol{x}_{i+1}\|}] \tag{4.2}$$

For path nudging, we choose an elastic band-inspired approach, which gives a physically interpretable way to tune the planning result. The original waypoints x_i^*

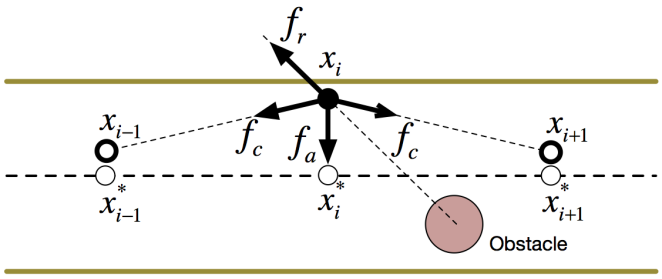


Figure 4.2: The definition of attractive, repulsive and contractive forces with discrete point samples.

are nudged laterally to x_i. The position of waypoints is determined by three artificial forces: attraction to the nominal path, contraction of the band, and repulsion from the obstacles (Figure 4.2). If a waypoint is nudged laterally such that it touches the corridor boundary, an adaptive repulsive force will be applied to balance the remaining force laterally, such that the boundary constraints are never violated:

$$\begin{aligned} f_a(x_i) &= \omega_a \cdot (x_i - x_i^*) \\ f_c(x_i) &= \omega_c \cdot \left(\frac{x_{i-1} - x_i}{\|x_{i-1} - x_i\|} + \frac{x_{i+1} - x_i}{\|x_{i+1} - x_i\|}\right) \\ f_r(x_i) &= \omega_r \cdot \frac{\delta[\rho(x_i) < \rho_t]}{[\rho_t - \rho(x_i)]^2} \cdot u\left[\frac{\partial \rho}{\partial x}(x_i)\right] \end{aligned} \tag{4.3}$$

where $\partial\rho(x)$ represents a certain potential function with respect to the state x (e.g., distance to obstacle), δ is the test function that returns 1 if the condition is true, and u is a function that takes the unit direction of a vector. A solution is reached when the sequence of x_i is found such that:

$$\underset{\{x_i\}}{\operatorname{argmin}} \sum_i \|f_a(x_i) + f_c(x_i) + f_r(x_i)\| \tag{4.4}$$

The path smoothing and nudging tasks are similar. They use the same waypoint representation with only one degree of freedom along the lateral direction with respect to the original waypoint x_i^*. Another common feature is that some of the cost terms (smoothness term in Equation 4.2 and contraction force in Equation 4.4) depend on not

only a single state, but also its neighboring (left/right) states, which introduces strong non-convexity.

While general nonlinear programming methods [Luenberger, 1973] can be applied to solve the optimization problem by directly changing x_i, the non-convexity of some cost terms, empirically, makes the solver suffer from robustness issues in terms of success rate and run-time. To avoid such numerical issues, our goal is thus to propose a graph-search-based method in the next section to numerically approximate the continuous optimization result with bounded runtime and a resolution-complete global optimality property.

4.2 Edge-Augmented Graph Search

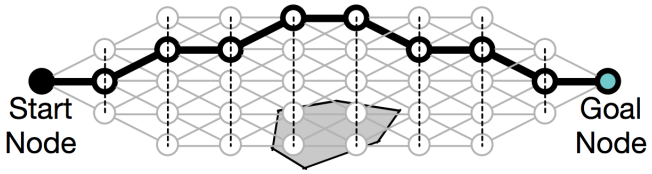
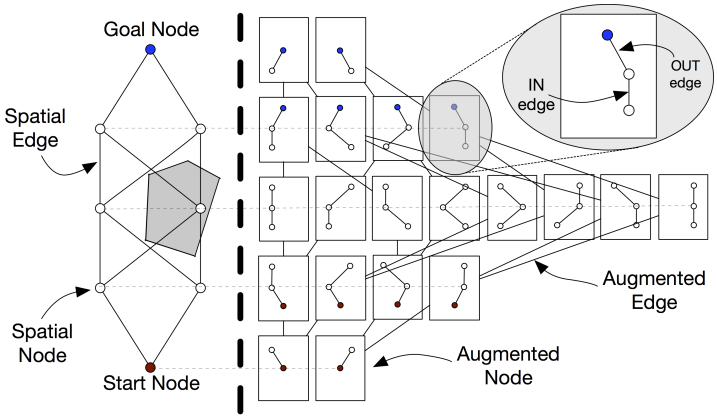


Figure 4.3: The spatial graph for numerical path smoothing/nudging.

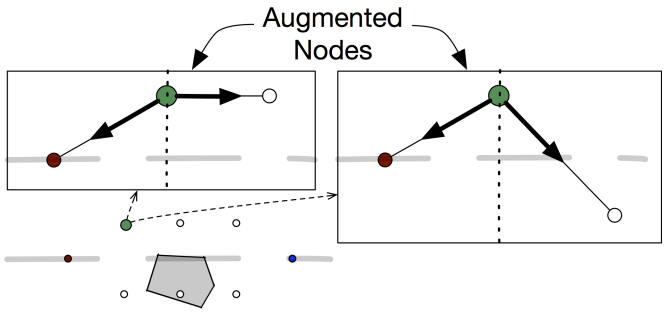
It is intuitive to create a discrete graph in the corridor environment, where the waypoints are further laterally sampled to create a spatial graph (Figure 4.3). Then the idea is to find a sequence of nodes in this graph that best approximate the continuous counterpart. The difficulty, however, is that for both path smoothing and path nudging, the existence of certain cost terms (smoothness term in Equation 4.2 and contractive force in Equation 4.4) makes it hard to directly express cost on the spatial graph, because each cost term is linked not just to a particular node, or two neighboring nodes (an edge), but three nodes (as shown in Figure 4.1 and 4.2). We therefore want to create a novel graph, the edge-augmented graph, that captures such information for planning.

三个点的cost需要转换为其他的形式

An edge-augmented graph is constructed in two steps. First, a traditional spatial graph is created by laterally sampling the waypoints along the nominal path (e.g., the



(a) Construct edge-augmented graph



(b) Zoomed in view of two augmented nodes from a spatial node

Figure 4.4: The construction of edge-augmented graph from spatial graph.

centerline), as shown in Figure 4.4. Spatial edges are evaluated for collision when constructing the elastic nodes, which removes infeasible edges through construction. Then, augmented nodes and augmented edges (Figure 4.4(a)) are composed by augmenting the spatial node with an IN edge and an OUT edge that connect the neighboring spatial nodes, and two elastic nodes are connected by an elastic edge. Suppose there are M incoming edges and N outgoing edges for a spatial node, then a total of $M \cdot N$ augmented nodes will be created for this spatial node and connected to its neighboring augmented nodes through augmented edges. Each augmented node has a corresponding spatial node as the center, and from the incoming/outgoing edges we can also infer the nodes connected to it (Figure 4.4(b)). Therefore, we are able to

conveniently calculate the smoothness term (path smoothing) and the contractive force (path nudging). Note that the spatial graph is constructed with predefined resolution, so the size of the edge-augmented graph is also predefined. Planning over this graph will return in deterministic runtime.

Planning on the graph is modeled as a single-source shortest-path (SSSP) problem. Similar to the spatial graph, the edges of the augmented graph also point monotonically ahead, and therefore formulate a directed acyclic graph (DAG). The Bellman-Ford algorithm (dynamic programming) can be used with $\mathcal{O}_{time}(|V| \cdot |E|)$. Topological sort can be used to search with a time complexity of $\mathcal{O}_{time}(|V| + |E|)$. In practice, the most time-consuming step is the evaluation of spatial graph's edges, in which collision checking/distance calculation is performed.

4.3 Constraint-satisfying Nominal Reference Generation

With the algorithm above, one can obtain a sequence of augmented nodes, which can be converted into a trace of linear spatial path (edges on the graph). However, there are two problems with this path.

The first problem is that the path is piecewise-linear (non-smooth), so that it is kinematically infeasible and does not contain useful curvature information, which is useful for speed profile generation in this section. Luckily, the LQR-based trajectory smoother explained in Section 3.3 provides an efficient method to obtain a kinematically feasible smooth path $\mathcal{P}$.

无曲率信息，而曲率信息用于生成速度是重要的信息

LQR方法可以生成运动学可行的平滑Path

Another problem is that no speed information is attached to $\mathcal{P}$ to formulate a reference[1] $\mathcal{R}$. This speed information does not necessarily account for moving objects, but there are several dynamic constraints that it must conform to (e.g., lateral and longitudinal accelerations) when adding a speed profile to a spatial trace. The focus of this section is to use an efficient algorithm to append speed information subject to certain speed constraints.

上述的平滑Path无速度信息，速度信息将由下面的算法进行快速生成

[1] See nomenclature for difference between reference and trajectory.

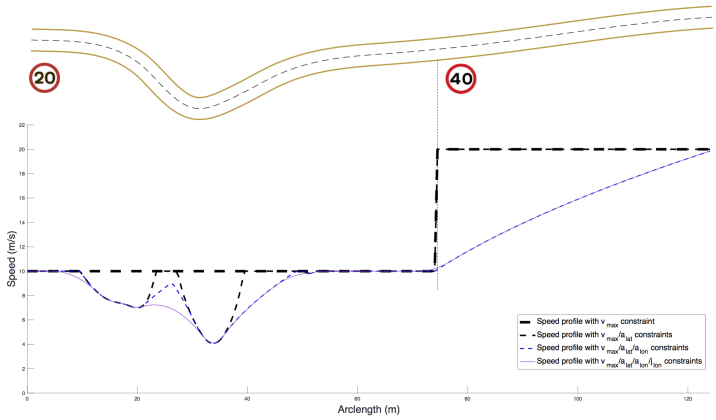


Figure 4.5: Generate speed profile with dynamic constraints.

The reference path $\mathcal{P}$ consists of a sequence of spatial states, such as position, heading, curvature and arc-length. For the purpose of speed profile generation, the curvature and arc-length sequence K and S are relevant. Several dynamic constraints will be imposed on the speed profile $\mathcal{V}$, which is a sequence of scalar speed values. Each constraint will be represented both in continuous and finite-difference forms to generate the speed profile, as illustrated in Figure 4.5.

For each iteration, the lateral acceleration a_{max}^{lat} constraint is applied first to bound the maximum speed at each point along the path:

$$\begin{aligned} &\text{continuous: } v \leq min\{\sqrt{\frac{a_{max}^{lat}}{\kappa}}, v_{max}\} \\ &\text{finite difference: } v_i \leq min\{\sqrt{\frac{a_{max}^{lat}}{\kappa_i}}, v_{max}\} \end{aligned} \tag{4.5}$$

Notice that $\sqrt{\frac{a_{lat}}{\kappa}}$ approaches/reaches singularity on low-curvature/straight segments as $\kappa \to 0$, which is the reason having v_{max} in the min function.

Though the lateral acceleration is capped, the speed profile is still unachievable since the longitudinal acceleration is not yet constrained (thin black dashed line in Figure 4.5). Preferred longitudinal acceleration a_{max}^{lon} and deceleration d_{max}^{lon} are therefore

applied:

$$\begin{aligned} \text{continuous: } & -d_{max}^{lon} \leq \dot{v} \leq a_{max}^{lon} \\ \text{finite difference: } & -d_{max}^{lon} \leq \frac{v_i^2 - v_{i-1}^2}{2|s_i - s_{i-1}|} \leq a_{max}^{lon} \end{aligned} \tag{4.6}$$

Excessive longitudinal jerk may be observed (see, for example, the blue dashed line in Figure 4.5). A maximum longitudinal jerk j_{max}^{lon} constraint can be further enforced. Its finite-difference representation must be deduced. Assume we can interpolate a speed function based on a quadratic polynomial of arc-length:

$$\begin{aligned} v &= \alpha \cdot s^2 + \beta \cdot s + \gamma \\ a &= 2\alpha \cdot sv + \beta \cdot v \\ j &= 2\alpha \cdot (v^2 + sa) + \beta \cdot a \end{aligned} \tag{4.7}$$

where s is the arc-length, v is the speed, by taking the first and second derivative w.r.t. time, we can further obtain the analytical form of the acceleration a and jerk j.

As shown in the function above, we have a constant jerk interpolation over the polynomial as long as all the coefficients α, β and γ can be calculated. Now, assuming that we have three (the previous, current and next) speed points v_{i-1}, v_i, v_{i+1} with their corresponding arc-length points s_{i-1}, s_i, s_{i+1}, they satisfy:

$$\begin{bmatrix} s_{i-1}^2 & s_{i-1} & 1 \\ s_i^2 & s_i & 1 \\ s_{i+1}^2 & s_{i+1} & 1 \end{bmatrix} \cdot \begin{bmatrix} \alpha_i \\ \beta_i \\ \gamma_i \end{bmatrix} = \begin{bmatrix} v_{i-1} \\ v_i \\ v_{i+1} \end{bmatrix} \tag{4.8}$$

Coefficients α_i, β_i and γ_i can all be analytically calculated by solving the linear equation above. Plug in the solved coefficients into Equation 4.7, we can solve uniquely for the numerical approximation of jerk j_i of the current point.

The goal then is to adjust the current speed point v_i such that the corresponding interpolated jerk j_i is bounded by j_{max}^{lon}:

$$\begin{aligned} \text{continuous: } & |\ddot{v}| \leq j_{max}^{lon} \\ \text{finite difference: } & |j_i| \leq j_{max}^{lon} \end{aligned} \tag{4.9}$$

Algorithm 1 iteratively modifies the speed profile taking all constraints into account at every cycle until the speed profile no longer changes. To enforce each constraint, the speed of the current point v_i is adjusted to meet the constraint values.

Algorithm 1 Generate constrained speed profile 速度生成算法

Require: Velocity profile $\mathcal{V}$ with maximum speed.
Ensure: Constrained velocity profile $\mathcal{V}^*$.

function LIMITLATERALACCELERATION($\mathcal{V}$, K, a_{max}^{lat}, v_{max})
  $N \leftarrow \mathcal{V}.length()$
  for $i \in [0 \rightarrow N)$ **do**
    $(v_i, \kappa_i) \in zip(\mathcal{V}, K)$
    $v_i \leftarrow MIN(\sqrt{\frac{a_{max}^{lat}}{\kappa_i}}, v_{max})$
  end for
  return $\mathcal{V}$
end function
function LIMITLONGITUDINALACCELERATION($\mathcal{V}$, S, a_{max}^{lon}, d_{max}^{lon})
  $N \leftarrow \mathcal{V}.length()$
  for $i \in [0 \rightarrow N-1)$ **do**
    $(v_{i-1}, s_{i-1}) \in zip(\mathcal{V}, S)$
    $(v_i, s_i) \in zip(\mathcal{V}, S)$
    $ds \leftarrow s_i - s_{i-1}$
    $v_i \leftarrow min(v_i, \sqrt{v_{i-1}^2 + 2 \cdot a_{max}^{lon} \cdot ds})$
  end for
  for $i \in [N-1 \rightarrow 0)$ **do**
    $(v_{i+1}, s_{i+1}) \in zip(\mathcal{V}, S)$
    $(v_i, s_i) \in zip(\mathcal{V}, S)$
    $ds \leftarrow s_{i+1} - s_i$
    $v_i \leftarrow min(v_i, \sqrt{v_{i+1}^2 + 2 \cdot d_{max}^{lon} \cdot ds})$
  end for
  return $\mathcal{V}$
end function
function LIMITLONGITUDINALJERK($\mathcal{V}$,j_{max})
  $N \leftarrow \mathcal{V}.length()$
  for $i \in [2 \rightarrow N-2]$ **do**
    $(v_{i-1}, s_{i-1}) \in zip(\mathcal{V}, S)$
    $(v_i, s_i) \in zip(\mathcal{V}, S)$
    $(v_{i+1}, s_{i+1}) \in zip(\mathcal{V}, S)$
    $j_i = \frac{2v_{i+1}}{s_{i-1}s_i - s_{i-1}s_{i+1} - s_i s_{i+1} + s_{i+1}^2} - \frac{2v_{i-1}}{s_{i-1}s_i + s_{i-1}s_{i+1} - s_i s_{i+1} - s_{i-1}^2} - \frac{2v_i}{s_{i-1}s_i - s_{i-1}s_{i+1} + s_i s_{i+1} - s_i^2}$
    if $j_i > j_{max}$ **then**
      $v_i \leftarrow -(j_{max} - \frac{2v_{i+1}}{s_{i-1}s_i - s_{i-1}s_{i+1} - s_i s_{i+1} + s_{i+1}^2} + \frac{2v_{i-1}}{s_{i-1}s_i + s_{i-1}s_{i+1} - s_i s_{i+1} - s_{i-1}^2})(\frac{s_{i-1}s_i}{2} - \frac{s_{i-1}s_{i+1}}{2} + \frac{s_i s_{i+1}}{2} - \frac{s_i^2}{2})$
    end if
    if $j_i < -j_{max}$ **then**
      $v_i \leftarrow (j_{max} + \frac{2v_{i+1}}{s_{i-1}s_i - s_{i-1}s_{i+1} - s_i s_{i+1} + s_{i+1}^2} - \frac{2v_{i-1}}{s_{i-1}s_i + s_{i-1}s_{i+1} - s_i s_{i+1} - s_{i-1}^2})(\frac{s_{i-1}s_i}{2} - \frac{s_{i-1}s_{i+1}}{2} + \frac{s_i s_{i+1}}{2} - \frac{s_i^2}{2})$
    end if
  end for
  return $\mathcal{V}$
end function
do
  $\mathcal{V}_0 \leftarrow \mathcal{V}$
  $\mathcal{V} \leftarrow LimitLateralAcceleration(\mathcal{V}, K, a_{max}^{lat}, v_{max})$
  $\mathcal{V} \leftarrow LimitLongitudinalAcceleration(\mathcal{V}, a_{max}^{lon}, d_{max}^{lon})$
  $\mathcal{V} \leftarrow LimitLongitudinalJerk(\mathcal{V}, j_{max})$
  $\mathcal{V}_f \leftarrow \mathcal{V}$
while $\|\mathcal{V}_f - \mathcal{V}_0\| > \epsilon$
$\mathcal{V}^* \leftarrow \mathcal{V}_f$

迭代进行求解

对于后续进行种子轨迹生成的时候，是否需要进行每一条种子轨迹都进行生成？［因为此处的约束条件只考虑到了极值］

Chapter 5

Maneuver Pattern Analysis

Topology information is important for motion planning. The existence of obstacles creates topological structures in the environment, which typically reflect distinctive maneuver patterns. The application of such knowledge helps focus the traditional trajectory planning (search / optimization) effort and if properly used, can also help enable higher-level behavioral reasoning capability.

In this chapter, we are primarily concerned with topological analysis (TA) for trajectory in a 3-D spatiotemporal space (2-D spatial space augmented with a time dimension), and using TA tools for maneuver pattern analysis.

5.1 Theoretical Background

Important concepts related to topology analysis is first be summarized in Section 5.1.1, followed by the physics-inspired analysis tools in Section 5.1.2 and its extension to the spatiotemporal space in Section 5.1.3. Many definitions and key results are borrowed from [Griffiths and College, 1999, Hatcher, 2002, Bhattacharya, 2012], leaving out a few rigorous definitions in algebraic topology such as chain complex, homology groups, de Rham cohomology group, etc. But this simplification will not hamper the description of the technical tools relevant for our purposes.

5.1.1 Concepts in Algebraic Topology

Topology Space: a topology on a set X is a collection, T, of subsets of X, containing both X and $\varnothing$, and closed under the operation of intersection and union. The tuple (X, T) is called a topological space S, and the elements of T are called open sets of the topological space.

Homology: homology is a mathematical concept for defining and counting the number of hole structures in a topological space.

Homotopy: a homotopy between two continuous functions f and g from a topological space X to a topological space Y is defined by a continuous function $H : X \times [0,1] \rightarrow Y$ from the product of the space X with the unit interval $[0,1]$ to Y such that, if $x \in X$ then $H(x,0) = f(x)$ and $H(x,1) = g(x)$. H is the continuous deformation of function f into g.

Homological Path: Two homological paths are co-terminal[1], and the boundary formed by connecting them tail-to-head does not contain any obstacle.

Homotopic Path: Two homotopic paths are co-terminal, and deformable from one to the other without intersecting any obstacle.

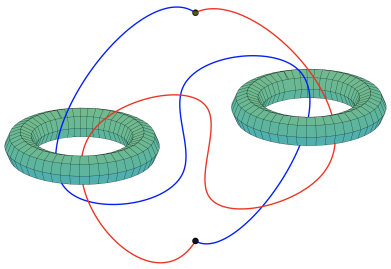


Figure 5.1: Illustration of two homological but non-homotopic trajectories.

[1]The terminology of *co-terminal* paths is borrowed from Jenkins [Jenkins, 1991] to describe two paths with the same starting point and destination point.

In Figure 5.1, trajectories $\mathcal{T}'$ (red) and $\mathcal{T}''$ (blue) are homological, since the loop formed by these two trajectories does not contain any one of the obstacle rings (imagine taking one obstacle ring away, after which the trajectory loop can be removed from the other ring without intersection by simply grasping it at one of the terminal points and pulling it through the center of the ring). However, they are not homotopic, since they cannot be deformed from one to the other when two obstacle rings exist at the same time. In general, the following lemma holds:

Lemma 5.1.1 *if two trajectories are homotopic, they are homological; the converse does not necessarily hold true.* 如果两条轨迹是同伦的，则它们是同源的；反之不一定成立

同伦等价 **Homotopy Equivalence:** two topological spaces X and Y are called homotopy-equivalent if there exist continuous functions $f : X \to Y$ and $g : Y \to X$ such that $g \circ f$ and $f \circ g$ are both homotopic to the identity map. f and g are called the homotopy equation, and X and Y can be homotoped to each other.

变形收缩 **Deformation Retract:** a subspace A is called a deformation retract of a topology space S if there exists a continuous function $f : S \times [0,1] \to S$ such that:

- $f(x,0) = x, \forall x \in X$
- $f(x,1) \in A, \forall x \in X$
- $f(a,t) = a, \forall a \in A, t \in [0,1]$

and f is a deformation retraction from S to A. A more intuitive test is whether the space S can be continuously shrunk and deformed to A without causing any cut or tear. If A is a deformation retract of S, then they are homotopy-equivalent.

Contractible space: a topological space S is called contractible if the identity map on it is homotopic to a constant map, in other words, taking every point in S to a fixed point in S. A more intuitive interpretation is that the space can be contracted continuously towards a point inside it.

5.1.2 Physics-Inspired Homology Identification

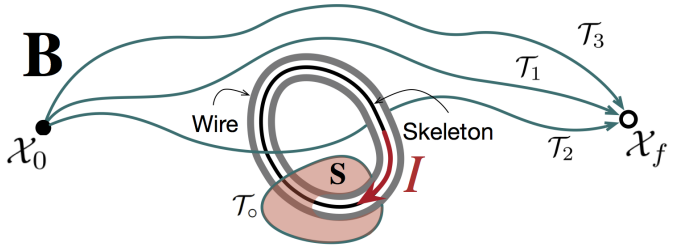


Figure 5.2: Energized wire and its generated magnetic field in the 3-D Euclidean space.

Biot-Savart Law: a steady current flowing through a wire $\mathcal{W}$ generates a magnetic field $\boldsymbol{B}$, the vector value of which defined at $\boldsymbol{r} \in \mathbb{R}^3$ is:

$$\boldsymbol{B}(\boldsymbol{r}) = \frac{\mu_0 \cdot I}{4\pi} \int_{\mathcal{W}} \frac{d\boldsymbol{l} \times (\boldsymbol{l} - \boldsymbol{r})}{\|\boldsymbol{l} - \boldsymbol{r}\|^3} \tag{5.1}$$

where I is the current in $\mathcal{W}$, $\boldsymbol{l}$ is a point on $\mathcal{W}$, and μ_0 is the magnetic constant, whose value is not significant for the purpose of homotopy analysis.

Ampere's Law: given a magnetic field $\boldsymbol{B}$ and a closed path $\mathcal{T}_\circ$, the line integral along $\mathcal{T}_\circ$ is proportional to the total current I passing through a surface $\boldsymbol{S}$ enclosed by $\mathcal{T}_\circ$

$$\oint_{\mathcal{T}_\circ} \boldsymbol{B} \cdot d\boldsymbol{l} = \mu_0 \cdot I \tag{5.2}$$

The laws of Biot-Savart and Ampere are dual theorems relating the energized looped wire $\mathcal{W}$ and the magnetic field generated. In Figure 5.2, regardless of the shape of the closed path $\mathcal{T}_\circ$, the integration of magnetic field $\boldsymbol{B}$ gives an Ampere invariant which is only relevant to the amount of current going through the closed surface $\boldsymbol{S}$ formed by $\mathcal{T}_\circ$. This property can be exploited to define the $\mathcal{H}$-signature for co-terminal trajectories like $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$ in Figure 5.2.

Imagine planning with one ring-like obstacle, which is energized by current I and generates a magnetic field $\boldsymbol{B}$. The homology-signature [Kim et al., 2012] is defined as the integration of the magnetic field along trajectory $\mathcal{T}$:

$$\mathcal{H}(\mathcal{T}) = \int_{\mathcal{T}} \boldsymbol{B} \cdot d\boldsymbol{l} \tag{5.3}$$

It can be shown that the homology-signature gives a numerical invariant to homological trajectories. In other words, trajectories $\mathcal{T}'$ and $\mathcal{T}''$ are homological if and only if their homology-signatures are equal. Note that while the trajectories are in the 3-D Euclidean space $\mathbb{R}^3$, this value can be used to detect trajectories in any 3-D space, including the spatiotemporal space $\mathbb{W}$ that will be explained in Section 5.1.3.

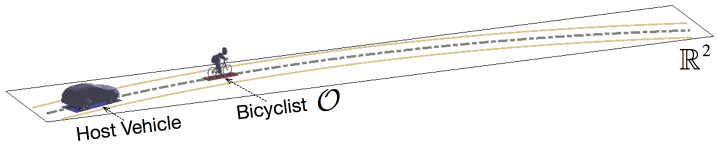
One important question to ask is: why the homology-signature inspired by electromagnetism theory can detect homology? As explained in [Bhattacharya, 2012], the differential 1-form (infinitesimal length/cost) captures the metric information about the underlying space. The homology-signature is a kind of differential 1-form that encodes topological information about the space. From a pure algebraic topology perspective [Bott and Tu, 2013], the differential 1-form (homology-signature) belongs to the De Rham cohomology group $H^1_{dR}(\mathbb{R}^D - \mathcal{O})$ for Euclidean space $\mathbb{R}^D$ punctured by the obstacle $\mathcal{O}$, and therefore it detects homology.

5.1.3 Topological Planning in 3-D Spatiotemporal Configuration Space

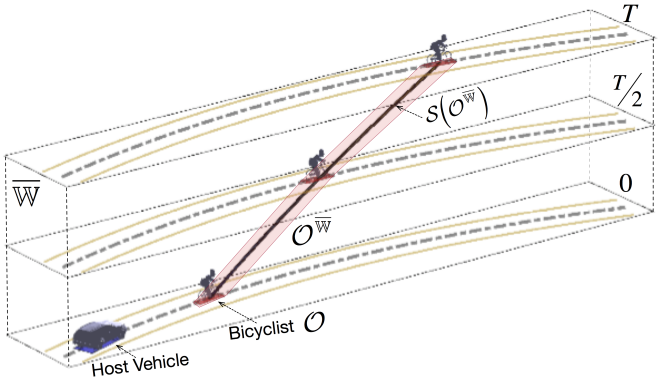
The APV operates in highway and urban environments, which are three-dimensional. The tilt and pitch of the APV on hilly roads changes the vehicle dynamics and shifts the center of gravity. While such factors must be compensated for control purposes, it is common practice to simply assume the workspace to be a 2-D plane and to leave out the details of the terrain for efficiency in planning.

The planning space of the APV henceforth becomes a 3-D spatiotemporal space $\mathbb{W} = [\mathbb{R}^2 \times \mathbb{T}]$ obtained by augmenting a 2-D planar space $\mathbb{R}^2$ with a time dimension $\mathbb{T}$. A bounded workspace $\overline{\mathbb{W}}$ is considered in practice due to the fixed planning horizon (T) at every cycle.

Properly responding to the surrounding objects is the primary goal of motion planning for urban autonomous driving. Taking a snapshot at any given time, an object can be represented by a 2-D polygon. Adding the temporal planning horizon (up to T) to a 2-D object creates a 3-D temporal object. Figure 5.3(a) shows an on-road driv-



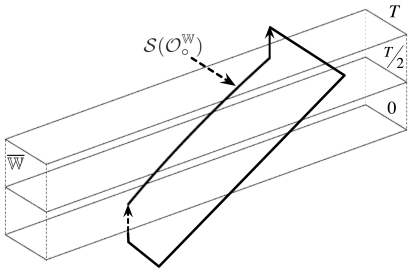
(a) Scenario with a single bicyclist on a single-lane road



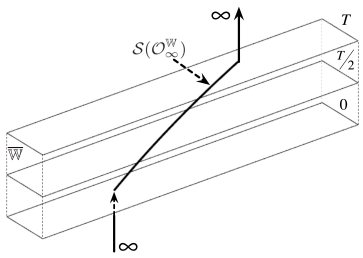
(b) Construct spatiotemporal planning space with prediction motion

Figure 5.3: The construction of 3-D spatiotemporal planning space.

ing scenario with a single bicyclist on a single-lane road. Figure 5.3(b) demonstrates that the bicyclist $\mathcal{O}$ is augmented spatiotemporally along its predicted trajectory, with $\mathcal{S}(\mathcal{O}^{\overline{\mathbb{W}}})$ being the skeleton of the temporal object $\mathcal{O}^{\overline{\mathbb{W}}}$ within $\overline{\mathbb{W}}$.



(a) Loop through arbitrary shape



(b) Loop through infinity

Figure 5.4: Create looped augmented temporal object for topological differentiation.

The temporal object $\mathcal{O}^{\overline{\mathbb{W}}}$ is genus[2]-0, and not able to incur different homologi-

[2]The genus of an object, an important concept in algebraic topology, is the number of the handles, or holes in its shape. An intuitive interpretation of genus is the maximum number of "cuts" that can be

cal/homotopic trajectories (nor is its skeleton $\mathcal{S}(\mathcal{O}^{\overline{\mathbb{W}}})$). In order to introduce such an ability, a temporal object $\mathcal{O}^{\overline{\mathbb{W}}}$ is augmented to become a looped structure in the unbounded spatiotemporal space to create genus-1 object $\mathcal{O}^{\mathbb{W}}$ in the space $\{x|x \in \mathbb{W}, x \notin \overline{\mathbb{W}}\}$. This can be achieved by appending either a finite-augmentation, giving $\mathcal{O}^{\mathbb{W}}_{\circ}$ (Figure 5.4(a)), or through an infinity-augmentation, giving $\mathcal{O}^{\mathbb{W}}_{\infty}$ (Figure 5.4(b)).

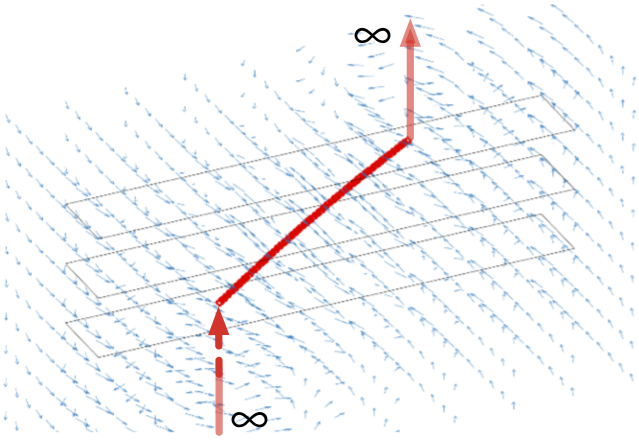


Figure 5.5: Create virtual magnetic field by applying virtual current in the augmented temporal object.

The augmented temporal object looped through infinity $\mathcal{S}(\mathcal{O}^{\overline{\mathbb{W}}}_{\infty})$ makes it easier to mathematically obtain the magnetic field (Figure 5.5) in a fashion that extends both ends of the skeleton to infinity in parallel to the time dimension. Within the loop, it becomes convenient to apply a virtual current flowing through, and inducing a magnetic field. If the skeleton of the augmented temporal object is made up of N straight line segments $\{\mathcal{W}_1, ..., \mathcal{W}_i, ..., \mathcal{W}_N\}$, where each line segment $\mathcal{W}_i$ is defined from point $\boldsymbol{l}_i$ to point $\boldsymbol{l}_{i+1}$, we can transform the original Biot-Savart Law to analytically calculate the vector field at r according to [Griffiths and College, 1999] without integration (which is very computationally efficient):

$$\mathbf{B}(\boldsymbol{r}) = \frac{\mu_0 \cdot I}{4\pi} \sum_{i=1}^{N} \int_{\mathcal{W}_i} \frac{d\boldsymbol{l} \times (\boldsymbol{l} - \boldsymbol{r})}{\|\boldsymbol{l} - \boldsymbol{r}\|^3} \tag{5.4}$$

performed on an object such that the cut object is still a connected object with full volume, e.g., in 3-D, a ball is genus-0, a loop is genus-1, etc.

where

$$\int_{\mathcal{W}_i} \frac{d\boldsymbol{l} \times (\boldsymbol{l} - \boldsymbol{r})}{\|\boldsymbol{l} - \boldsymbol{r}\|^3} = \frac{1}{\|\overrightarrow{d_i}\|^2}\left(\frac{\overrightarrow{d_i} \times \overrightarrow{q_i}}{\|\overrightarrow{q_i}\|} - \frac{\overrightarrow{d_i} \times \overrightarrow{p_i}}{\|\overrightarrow{p_i}\|}\right) \tag{5.5}$$

when

$$\begin{aligned} \overrightarrow{p_i} &= \boldsymbol{l_i} - \boldsymbol{r} \\ \overrightarrow{q_i} &= \boldsymbol{l_{i+1}} - \boldsymbol{r} \\ \overrightarrow{d_i} &= \frac{(\boldsymbol{l_{i+1}} - \boldsymbol{l_i}) \times (\overrightarrow{p_i} \times \overrightarrow{q_i})}{\|\boldsymbol{l_{i+1}} - \boldsymbol{l_i}\|^2} \end{aligned} \tag{5.6}$$

Note that for the last ($\mathcal{W}_N$) segment, these equations are changed since $\boldsymbol{l}_{N+1}$ goes to infinity:

$$\int_{\mathcal{W}_N} \frac{d\boldsymbol{l} \times (\boldsymbol{l} - \boldsymbol{r})}{\|\boldsymbol{l} - \boldsymbol{r}\|^3} = \frac{1}{\|\overrightarrow{d_N}\|^2}\left(\overrightarrow{d_N} \times \overrightarrow{n_N} - \frac{\overrightarrow{d_N} \times \overrightarrow{p_N}}{\|\overrightarrow{p_N}\|}\right) \tag{5.7}$$

with $\overrightarrow{n_N}$ being the unit vector in the direction of line segment $\mathcal{W}_{N-1}$, and

$$\begin{aligned} \overrightarrow{p_N} &= \boldsymbol{l_{N-1}} - \boldsymbol{r} \\ \overrightarrow{d_N} &= \overrightarrow{n_N} \times (\overrightarrow{p_N} \times (\overrightarrow{p_N} + \overrightarrow{n_N})) \end{aligned} \tag{5.8}$$

By symmetry, we can obtain the integration result for the first line segment ($\mathcal{W}_1$) as well. Once the magnetic field is obtained, the $\mathcal{H}$-signature can be calculated by Equation 5.3 given trajectory $\mathcal{T}$.

As explained in Section 5.1, homotopic trajectory is also homological, but the converse does not necessarily hold true. An important question to ask is: why do we detect trajectory homology (weaker) rather than homotopy (stronger)? First, detecting homology is numerically straightforward with the homology-signature, and its invariance property will be further exploited in Section 5.2. Another reason is that the topological information we care about in the context of on-road driving has to do exclusively with swerving left or right around objects, and can be performed in a per-object fashion, in which homology and homotopy are equivalent.

However, it is also important to realize that such topological information is only part of the story for our ultimate goal, which is to distinguish between distinct maneuver patterns. In a stationary environment, topological information may be sufficient to distinguish among maneuver patterns, as shown in Section 5.2. On the other hand,

if multiple moving objects are involved, while homology/homotopy helps distinguish swerving left vs. right, it does not directly identify staying behind a particular object, nor does it handle complications like the sequence according to which to swerve around different obstacles, which will be addressed in Section 5.3.

5.2 Graph Segmentation-based Maneuver Pattern Identification

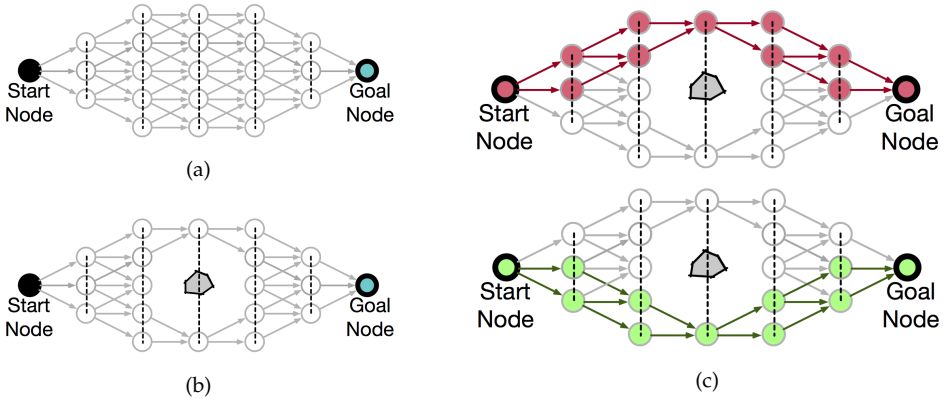


Figure 5.6: Construct DAG to perform graph segmentation-based topological analysis.

In this section, we consider the graph-based path planning problem in a simplified environment, where only static obstacles are included for consideration. The DAG will be constructed in a fashion that suits a corridor-like maneuver environment, similar to that shown in Figure 4.3. We start by removing infeasible edges from the DAG, i.e., we obtain Figure 5.6(b) from Figure 5.6(a). Rather than finding a minimum-cost route on this graph, the goal is to segment this graph into multiple topologically distinct sub-graphs to facilitate later planning within each sub-graph as in Figure 5.6(c). There are a few symbols used below:

- $\Delta^{\mathcal{H}}$: the homology-signature incremental w.r.t. a single object.
- $\Delta^{\boldsymbol{\mathcal{H}}}$: the homology-signature incremental vector w.r.t. all objects in consideration.
- $\mathcal{H}$: the homology-signature of a node w.r.t. a single object.

- $\boldsymbol{\mathcal{H}}$: the homology-signature vector of a node w.r.t. all objects in consideration.
- $S^{\mathcal{H}}$: the homological-signature set that keeps track of the homology-signature combinations (unique $\boldsymbol{\mathcal{H}}$) of topologically different trajectories for each node.

The first step is to understand the number of topologically distinct routes to reach the goal node for each node in this graph. Supposing we have M active obstacles for topology analysis, a homology-signature incremental $\Delta^{\mathcal{H}}$ can be calculated for each obstacle. Then for each edge in the graph, we can assemble a M-dimensional homology-signature incremental vector $\Delta^{\boldsymbol{\mathcal{H}}}$ along this edge (trajectory piece):

$$\Delta^{\boldsymbol{\mathcal{H}}} = [\Delta_1^{\mathcal{H}} \cdots \Delta_M^{\mathcal{H}}]^T \tag{5.9}$$

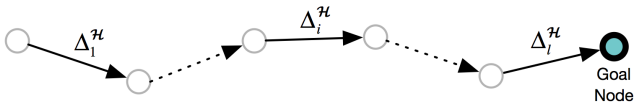


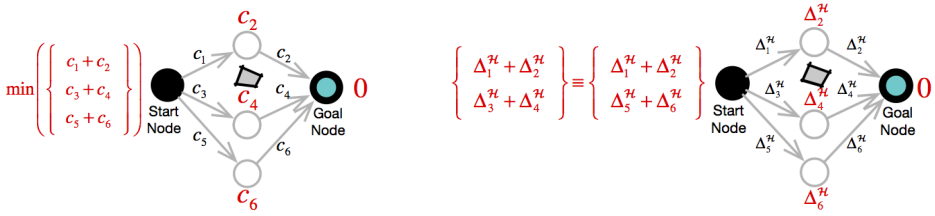
Figure 5.7: The homology-signature vector incrementals $\Delta^{\boldsymbol{\mathcal{H}}}$ along a given graph path.

For any given path on DAG composed by l edges (Figure 5.7), the homology-signature vector $\boldsymbol{\mathcal{H}}$ can be calculated:

$$\boldsymbol{\mathcal{H}} = \sum_{i=1}^{l} \Delta_i^{\boldsymbol{\mathcal{H}}} = [\mathcal{H}_1 \cdots \mathcal{H}_M]^T \tag{5.10}$$

Each unique $\mathcal{H}$-signature vector corresponds to a topologically distinct trajectory from the first point. We use the homological-signature set $S^{\boldsymbol{\mathcal{H}}} = \{\boldsymbol{\mathcal{H}}\}$ to keep track of the number of topologically different trajectories for each node.

The naive approach to generate the set $S^{\boldsymbol{\mathcal{H}}}$ is to expand all the path routes on the graph from the start node, and calculate $\boldsymbol{\mathcal{H}}$ along each and every route to investigate how many topological patterns exist starting from the current node. However, this will be an exponential algorithm, which is inefficient for any reasonable-sized problem. Instead, we draw inspiration from backward-induction-based dynamic programming [Dreyfus and Law, 1977] to perform graph analysis. Dynamic programming (Figure 5.8(a)) calculates optimal cost-to-go from current node to goal node with backward



(a) Dynamic programming (b) Topology induction

Figure 5.8: Comparison of backward propagation routines in dynamic programming (optimal cost-to-go) and topology induction (homology-signature vector).

induction. A similar induction method is used to propagate topological information over the graph, but replacing the optimal cost-to-go with the homology-set-to-go $S^{\mathcal{H}}$ (Figure 5.8(b)). This set keeps track of the unique $\boldsymbol{\mathcal{H}}$ (an $\boldsymbol{\mathcal{H}}$ being unique if at least one dimension is different from all other vectors in the set), which implies distinct homological routes achievable from a node to reach the goal node.

Let the current node be $\boldsymbol{n}^c$. It has l outgoing edges $\{\boldsymbol{e}_1, \cdots, \boldsymbol{e}_l\}$, each with a $\mathcal{H}$-signature vector incremental $\{\Delta_1^{\mathcal{H}}, \cdots, \Delta_l^{\mathcal{H}}\}$, that connects to l nodes $\{\boldsymbol{n}_1^o, \cdots, \boldsymbol{n}_l^o\}$, whose corresponding $\boldsymbol{\mathcal{H}}$ sets are $\{S^{\mathcal{H}}(\boldsymbol{n}_1^o), \cdots, S^{\mathcal{H}}(\boldsymbol{n}_l^o)\}$. The set $S^{\mathcal{H}}(\boldsymbol{n}^c)$ can be determined by a double union operator:

$$S^{\mathcal{H}}(\boldsymbol{n}^c) = \bigcup_{i=1}^{l} \bigcup_{\hat{\boldsymbol{\mathcal{H}}} \in S^{\mathcal{H}}(\boldsymbol{n}_i^o)} (\Delta^{\mathcal{H}_i} + \hat{\boldsymbol{\mathcal{H}}}) \tag{5.11}$$

Note that in Equation 5.11, the set of the current node can be uniquely determined by its outgoing nodes, which leads to a backward induction procedure. This is possible since each $\mathcal{H}$-signature is only relevant to the topology of the trajectory. The size of the set-to-go is curtailed by the topologically distinct patterns achievable in the search space (graph). For example, in Figure 5.8(b), there is only one object, so the $\boldsymbol{\mathcal{H}}$ vector has only one element. The edges of the two bottom routes from start to goal nodes each have different $\mathcal{H}$-signature increments, but since these two routes belong to the same homological class, it is guaranteed that $\Delta_3^{\mathcal{H}} + \Delta_4^{\mathcal{H}} \equiv \Delta_5^{\mathcal{H}} + \Delta_6^{\mathcal{H}}$. The backward induction proceeds from the goal node $\boldsymbol{n}^{goal}$ until the start node $\boldsymbol{n}^{start}$. At the end of

the process, each node has the set $S^{\mathcal{H}}$. Now, the second objective is to identify the regions that are subject to particular topological classes.

Each $\boldsymbol{\mathcal{H}}$ in the set $S^{\mathcal{H}}(\boldsymbol{n}^{start})$ indicates a unique homological path from start node $\boldsymbol{n}^{start}$ to the goal node. We pick a particular $\boldsymbol{\mathcal{H}}$ from the start, corresponding to a particular class tag, then perform a forward propagation to tag to all the nodes that are reachable subject to this homological constraint. Suppose a node has a tag, and its corresponding homology-signature vector of a node is $\boldsymbol{\mathcal{H}'}$. We iterate through all the outgoing edges $\boldsymbol{e'}$ and its connecting node $\boldsymbol{n}'$, and propagate this class tag if:

$$\boldsymbol{\mathcal{H}'} - \Delta^{\mathcal{H}}(\boldsymbol{e}') \in S^{\mathcal{H}}(\boldsymbol{n}') \tag{5.12}$$

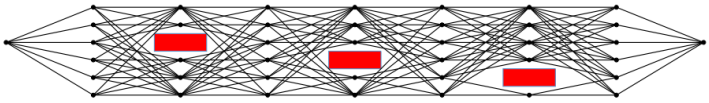
This procedure is continued until the goal node is encountered. Then all the nodes visitable subject to this class are generated. This process is further performed for other vector in $S^{\mathcal{H}}(\boldsymbol{n}^{start})$ to identify regions for all classes.

The final result is a segmented graph for each of these labels. Marking them with distinct colors will generate plots like Figure 5.6(c). A more involved situation is demonstrated in Figure 5.9. This provides a convenient way for the planner to obtain high-level understanding of the maneuver space, and can be used as a pre-processing step to focus the graph search effort. The method described in this section can be used to serve as an additional step before applying the edge-augmented graph search approach described in Section 4.4 to limit the plan within a certain topological class.

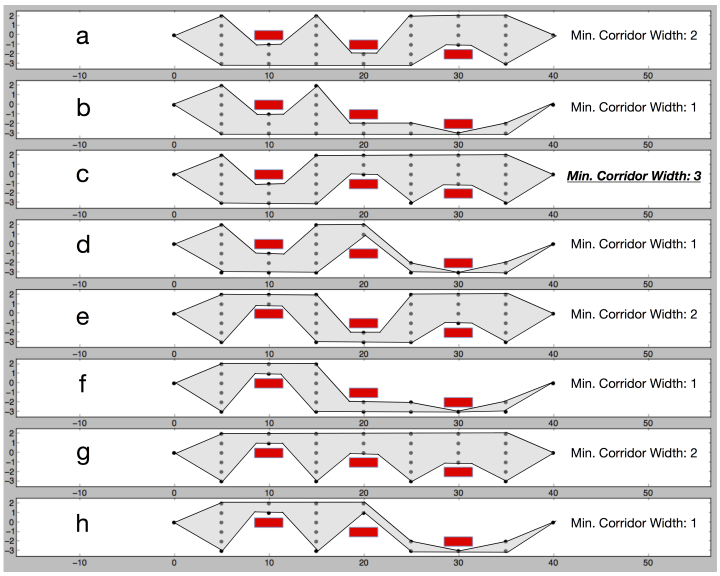
5.3 Sampling-based Maneuver Pattern Identification

Maneuver pattern analysis in Section 5.2 is limited to a 2-D corridor-like spatial space. In this section, analysis is performed in the spatiotemporal space, where the predicted motions of moving objects are taken into account.

A similar DAG graph can be constructed in the spatiotemporal space (Figure 5.10(a)). However, it is difficult to depend solely on the homology-signature backward induction routine discussed in Section 5.2 for the full analysis, because:



(a) Spatial DAG and three objects

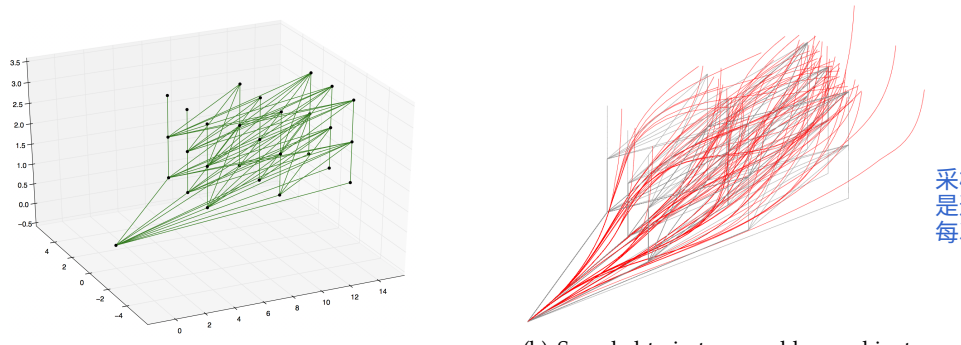


(b) Topologically distinct regions

Figure 5.9: Graph segmentation-based topological analysis for a three-object scenario.

- In the planning on a 2-D spatial DAG graph (Figure 4.3), we can always put a virtual goal node in the corridor, because the planning horizon of a path can always be hallucinated with a fixed node. But for a 3-D spatiotemporal DAG graph (Figure 5.10(a)), while it can be predetermined to plan up to a fixed time horizon T seconds, the exact goal point should not be hallucinated, but should be determined by the planning itself. Without a fixed goal node, the homology-signature is undefined.

- For spatiotemporal analysis, knowledge other than topological information must be used to characterize the maneuver pattern, such as the region information and the sequence information. This information is independent from and does not fit into homology-signature backward induction process.



(a) Linear DAG in spatiotemporal space

(b) Sampled trajectory pool by combinatorial iteration and LQR-based smoothing

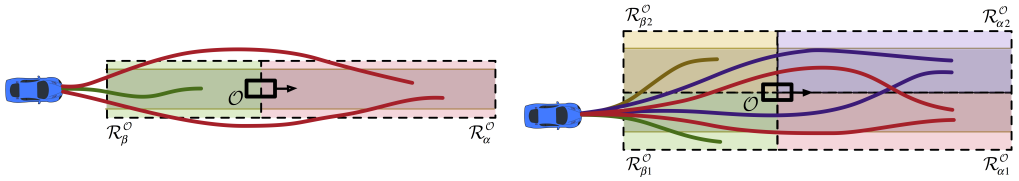
采样池是否为固定不变的，如果是这样的话，就可以不需要进行每次迭代了

Figure 5.10: Construct spatiotemporal trajectory sampling pool.

To this end, after constructing a coarse (linear-edge) 3-D DAG (Figure 5.10(a)), rather than performing the topology induction, we sample a pool of full T-second trajectory candidates by enumerating possible route sequences, and then perform spatiotemporal pattern analysis to generate sample trajectories (Figure 5.10(b)). In the on-road planning setting, infeasible trajectories that cause collision with any objects, or deviate too much from the spatial corridor, will be removed. The number of trajectories is exponential w.r.t. the number of sampling layers, but their generation and analysis are straightforwardly parallelizable.

The central task of maneuver pattern analysis on a trajectory sample pool is to differentiate trajectories that belong to distinct maneuvers. In the context of on-road driving, we define three main sources of distinction:

- **Region-based Distinction:** distinguish the trajectories that terminate in different regions at the end of the T-second planning horizon. T秒后的区域
- **Topology-based Distinction:** for maneuvers that involve swerving around an object, homology information helps determine on which side the maneuver is performed. 针对绕行动作，同源分析可以区分执行的动作是哪一个方向
- **Sequence-based Distinction**: if overtaking multiple objects is involved, the order of overtaking further give us certain cues. 多物体超车的先后顺序



(a) Two regions for single lane (b) Four regions for double lane

Figure 5.11: Region-based and topology-based distinctions for on-road driving.

区域

Region-based Distinction In the context of on-road driving, where the effective spatial planning domain is just a corridoor, it is useful to define the concept of an overtaking maneuver by defining an overtaking region. Intuitively, an overtaking maneuver describes the behavior of an APV moving from a following position to a leading position w.r.t. an object within a limited time horizon (T seconds) within an active planning corridor (the union of two polygonal regions $\mathcal{R}_\alpha^{\mathcal{O}}$ and $\mathcal{R}_\beta^{\mathcal{O}}$ in Figure 5.11(a)). In this corridor, given an object's predicted motion trajectory, two regions $\mathcal{R}_\alpha^{\mathcal{O}}$ and $\mathcal{R}_\beta^{\mathcal{O}}$ can be separated by the last known state on the predicted trajectory (the black object polygon in Figure 5.11(a)). If a planned trajectory terminates in $\mathcal{R}_\alpha^{\mathcal{O}}$, we say it is an overtaking trajectory; if in $\mathcal{R}_\beta^{\mathcal{O}}$, it is a following trajectory.

Similarly, the concept of overtaking can be extended to a two-corridor environment (e.g., considering the possibility of overtaking through lane-change.) In Figure 5.11(b), four regions can be distinguished: current-lane overtaking $\mathcal{R}_{\alpha 1}^{\mathcal{O}}$, current-lane following $\mathcal{R}_{\beta 1}^{\mathcal{O}}$, target-lane overtaking $\mathcal{R}_{\alpha 2}^{\mathcal{O}}$ and target-lane following $\mathcal{R}_{\beta 2}^{\mathcal{O}}$.

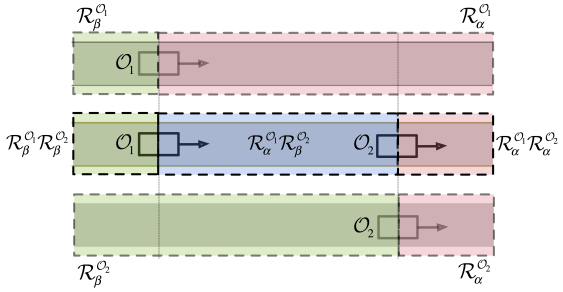


Figure 5.12: Identify gaps between moving vehicles through the combinations of object-associated regions.

When multiple objects are considered, the combination of terminal-regions with respect to each object naturally enriches the maneuver's descriptive capability. For example, in the highway scenario, this combination naturally identifies gaps between cars in the neighboring lane (middle lane in Figure 5.12).

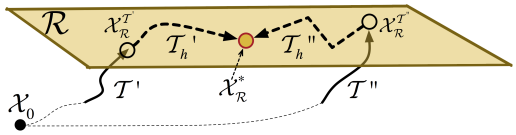


Figure 5.13: Construct helper trajectory to detect pseudo-homological trajectories.

Topology-based Distinction Trajectories with the same terminal-regions are not differentiated by the region-based distinction, but they do not necessarily coincide at the same terminal point. Therefore, the original definition for homology-signature (Equation 5.3) must be adapted. We relax the co-terminal requirements by defining pseudo-homology for trajectories that terminate in the same 2-D spatial region represented by a polygon (Curves of the same color in Figure 5.13).

In Figure 5.13, a point is the retract of a 2-D spatial region $\mathcal{R}$, also referred to as the representative point $\mathcal{X}_{\mathcal{R}}^*$ of $\mathcal{R}$. Note that $\mathcal{R}$ does not have to be a convex shape, and the exact location of $\mathcal{X}_{\mathcal{R}}^*$ is insignificant as long as it is contained by $\mathcal{R}$. Then we can define pesudo-homology.

Pseudo-Homology: two trajectories $\mathcal{T}'$ and $\mathcal{T}''$ are pseudo-homological if they both start from the same state and end in the same path-connected region $\mathcal{R} \in \mathbb{R}^2$, that can be extended from where they terminate on $\mathcal{R}$ to the representative point $\mathcal{X}_{\mathcal{R}}^*$ with helper trajectories $\mathcal{T}_h'$ and $\mathcal{T}_h''$, and

$$\mathcal{H}(\mathcal{T}' + \mathcal{T}_h') = \mathcal{H}(\mathcal{T}'' + \mathcal{T}_h'') \tag{5.13}$$

The helper trajectories stay within $\mathcal{R}$, and can be of arbitrary shape (Figure 5.13).

Revisiting Figure 5.11, trajectories may reach certain regions by taking topologically different routes. All the topologically distinctive trajectories are depicted in the same

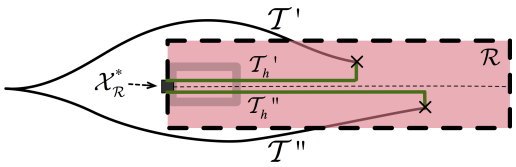


Figure 5.14: Detect pseudo-homology for trajectories that terminate in the same on-road corridor region.

color. In addition, efficient ways to determine the representative point and construct helper trajectories for each identified region are required by the algorithm. Taking advantage of the fact that the region $\mathcal{R}$ (Figure 5.14) is constructed from corridor-like on-road lanes, we place $\mathcal{X}_{\mathcal{R}}^*$ on the centerline, close to the region boundary near the object itself. The helper trajectories are then constructed by connecting from the location where the trajectory terminates in the region, to the projected point perpendicular to the centerline, and further moving along the centerline to reach $\mathcal{X}_{\mathcal{R}}^*$. The procedure to determine the pseudo-homology is summarized in Algorithm 2:

Algorithm 2 Determine pseudo-homology

Require: Two trajectories $\mathcal{T}'$ and $\mathcal{T}''$
Ensure: Correct judgment of trajectories' pseudo-homology
  IDENTIFY a spatial region $\mathcal{R} \in \mathbb{R}^2$ within $\overline{\mathbb{W}}$ at time T.
  CHECK if the end states $\mathcal{X}^{\mathcal{T}'}$ and $\mathcal{X}^{\mathcal{T}''}$ are in $\mathcal{R}$.
    IF no, **RETURN** false
  RETRACT region $\mathcal{R}$ to a representative point $\mathcal{X}_{\mathcal{R}}^*$
  CONSTRUCT helper trajectories $\mathcal{T}_h'$ and $\mathcal{T}_h''$ that connect $\mathcal{X}_{\mathcal{R}}^{\mathcal{T}'}$ and $\mathcal{X}_{\mathcal{R}}^{\mathcal{T}''}$ to $\mathcal{X}_{\mathcal{R}}^*$
  CALCULATE the $\mathcal{H}$ function of $\mathcal{T}' + \mathcal{T}_h'$ and $\mathcal{T}'' + \mathcal{T}_h''$
    IF $\mathcal{H}(\mathcal{T}' + \mathcal{T}_h') \neq \mathcal{H}(\mathcal{T}'' + \mathcal{T}_h'')$ **RETURN** false
    ELSE RETURN true

Sequence-based Distinction It is common in planning to generate trajectories that overtake multiple objects. In this situation, only knowing the topological information w.r.t each object is not sufficient to distinguish some important maneuver patterns. For example, in Figure 5.15(a), the two trajectories reflect two distinct maneuver patterns: the upper one is more conservative by a delayed bicyclist overtaking after passing a parked car, while the lower one is more aggressive by overtaking the bicyclist through

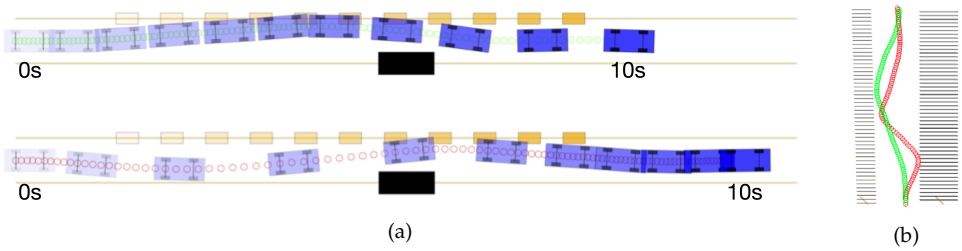


Figure 5.15: Two pseudo-homological trajectories demonstrate distinct maneuver patterns with different overtaking sequencing.

the closing gap between the bicyclist and the parked car. However, from a pure topological perspective, they are equivalent by being pseudo-homological/homotopic in the 3-D spatiotemporal space (Figure 5.15(b)).

To be aware of the different overtaking sequences, in the trajectory evaluation process, we need to forward-simulate simultaneously the APV and objects, keep track of the time-stamp when an overtaking maneuver is accomplished w.r.t. to each object, and then sort against the time-stamp, which will give a sequence description. If two overtaking maneuvers are accomplished at the same time (strictly speaking, this never happens in the real world, but it is possible in the planning world due to finite sampling of a trajectory). For tie breaking, we assume there is a secondary sorting criterion, which can simply be an arbitrary index number of the object itself.

For a given trajectory, a pattern distinction tree can be constructed. In Figure 5.16, four objects are actively considered for pattern distinctions. Objects $\mathcal{O}_1$, $\mathcal{O}_2$ and $\mathcal{O}_4$ have two regions as in Figure 5.11(a), while object $\mathcal{O}_3$ has four regions as in Figure 5.11(b). For any α (overtaking) region, there are two pseudo-homological alternatives: swerve left $\mathcal{H}_l$ and swerve right $\mathcal{H}_r$. For any β (following) region, there is only one pseudo-homological trajectory.

With this tree, we apply the region-based and topology-based distinction routine for each object to "paint" the routes that a particular trajectory $\mathcal{T}$ belongs to. For example, $\mathcal{T}$ will overtake object $\mathcal{O}_1$ on the left, follow object $\mathcal{O}_2$, overtake object $\mathcal{O}_3$ from the left to the target lane (in a lane change), while overtaking $\mathcal{O}_4$ on the right.

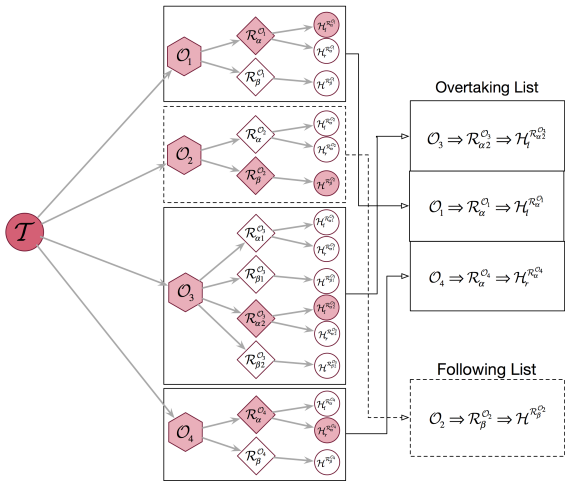


Figure 5.16: Example of constructing maneuver distinction tree of four objects.

After painting the distinction tree, we put all the objects on which an overtaking maneuver was performed in an "Overtaking List", and sort them based on the timestamp (using the object index as a secondary criterion for tie breaking) reflecting when the overtaking maneuver is accomplished. Objects that involve only a following maneuver are put in a "Following List" and get sorted based on its index. To concisely represent the result of sampling-based maneuver pattern analysis, we represent the final pattern by stacking the overtaking list on top of the following list:

$$\mathcal{L} = \begin{bmatrix} \mathcal{L}_3 \\ \mathcal{L}_1 \\ \mathcal{L}_4 \\ \mathcal{L}_2 \end{bmatrix} = \begin{bmatrix} \mathcal{O}_3 \rightarrow \mathcal{R}_{\alpha 2}^{\mathcal{O}_3} \rightarrow \mathcal{H}_l^{\mathcal{R}_{\alpha 2}^{\mathcal{O}_3}} \\ \mathcal{O}_1 \rightarrow \mathcal{R}_{\alpha}^{\mathcal{O}_1} \rightarrow \mathcal{H}_l^{\mathcal{R}_{\alpha}^{\mathcal{O}_1}} \\ \mathcal{O}_4 \rightarrow \mathcal{R}_{\alpha}^{\mathcal{O}_4} \rightarrow \mathcal{H}_r^{\mathcal{R}_{\alpha}^{\mathcal{O}_4}} \\ \mathcal{O}_2 \rightarrow \mathcal{R}_{\beta}^{\mathcal{O}_2} \rightarrow \mathcal{H}^{\mathcal{R}_{\beta}^{\mathcal{O}_2}} \end{bmatrix} \tag{5.14}$$

Supposing each route has a unique identifier, the full pattern will have a unique identifier by concatenating the identifiers of each route sequentially. Then trajectories can be grouped together, which concludes the tactical maneuver pattern discovery process. Note that while the number of possible patterns generally grows exponentially as the number of objects increases, the runtime of our planner is linear w.r.t. the number of trajectories. So if we have a reasonably sized sample pool (e.g., less than 10,000), the computation is manageable.

5.4 Choosing Maneuver Pattern

In Section 5.2 and 5.3, maneuver patterns are identified in the form of graph segmentation regions and groups of trajectory samples, respectively. They both provide a high-level segmentation of the planning space. In this section, we make a few points regarding how the segmentation should be evaluated and selected.

In graph-segmentation-based pattern analysis, the graph regions are some portion of the path planning corridor on the DAG. If all static objects are simply treated as blockages, the optimal graph segmentation can be selected primarily according to the maximum narrowest passage width, which is a good indicator of traversability. For example, in Figure 5.9, region (c) is chosen for this reason. Other metrics linked to specific objects can also be designed.

空间图评估策略

In the sampling-based pattern analysis, trajectory samples are grouped by their pattern. While the cost function to score trajectories will be developed in Chapter 6, it is often impractical[3] to evaluate all the sampled trajectories (often numbering in the thousands) in this phase. Therefore, some high-level metrics that can be calculated quickly help, e.g., the internal metrics like the average/minimum/maximum accelerations.

Sometimes, the metrics of the current trajectory discussed above are not the only important factor to determine the choice of pattern. Another important factor is pattern consistency, which is achieved by persisting with a particular pattern unless there are drastic environmental changes. For that, we also need a measure of the degree of difference between the pattern selected for the last cycle and a given pattern in the sampled trajectory from the current cycle. Since an object may disappear or show up for the first time, the first step is to remove those object that exist in only one cycle. Then for each common object, we can compare and count the difference in region, in homotopy and in sequence between cycles (For graph segmentation-based analysis, only the homotopy difference may exist).

模式的保持

评估当前模式和上一模式的差异性

[3]Trajectory evaluation can be computationally expensive, especially when multiple terms are evaluated with distance calculations.

There are two additional notes. First, in this section, we have made an implicit assumption, which is that a given object, once detected, will have a unique index, and won't change. Second, the main contribution is two novel ways to segment the planning space, which emplaces high-level segmentation in the traditional search space. With sufficient manual driving logs, it is even possible to develop more features, even directly use the image space features (from the raw perception), to train a classifier to select the right maneuver pattern in real-time. But this is out of the scope of this thesis. The overall planning flow can be summarized as:

1. Create search space (Construct spatial DAG or spatiotemporal sampling).
2. Trim search space (Remove infeasible edges in spatial DAG or remove infeasible trajectories from sample).
3. Perform search space segmentation (Topology back-induction or grouping based on region-based, topology-based and sequence-based distinctions.)
4. Choose the segmentation based on some criterion.

Chapter 6

Focused Trajectory Optimization with Constraints

In Chapter 4 and 5, we proposed novel techniques that combine topological analysis and graph search to perform pattern identification and path/trajectory planning. Taking advantage of the smoothing procedure in Chapter 3, we can further obtain a model-feasible smooth trajectory that roughly encodes how the robot should behave subject to the selected maneuver pattern. However, there are a few limitations to the trajectory plan generated so far:

- The trajectory so far can be regarded as globally resolution-complete optimal, but subject to sampling suboptimality, in other words, not locally optimal.
- The trajectory so far is a result of evaluating the LQR-smoothing routine in Chapter 3, after which the collision-free guarantee promised by the graph-search planning result (e.g, piece-wise linear edge plans) may have been voided.
- There is a lack of cost terms linked to internal state/control, which plays a vital part in shaping the ultimate trajectory, in the cost function designed for both maneuver pattern selection and graph-search planning.

- The trajectory so far is model-feasible. However, it is important to recognize that model-feasibility is a necessary but not sufficient condition for execution-feasibility. Many state/control constraints were not enforced in the model differential equations.

The goal of this chapter is to review and adapt the well-studied iterative linear quadratic regulator (iLQR, a.k.a. sequential-LQR, first-order Differential Dynamic Programming) [Todorov and Li, 2005, Tassa et al., 2014, Jacobson and Mayne, 1970, van den Berg, 2016, Li and Todorov, 2004a, Jacobson, 1968] in the APV on-road driving application. The trajectory generated so far will be used as a good setting-off point to determine "which local minimum to choose", and to use an adapted trajectory optimization method to address the following challenging problem: determine an execution-feasible and locally optimal trajectory that is subject to a variety of state/control and maneuver pattern constraints.

6.1 Relationship between Planning & Control

The boundary between planning and control can be vague. In our context, two planning-control hierarchical paradigms are discussed: the tracker-dependent approach and the open-loop approach. The planners in both approaches are capable of generating the trajectory. The subtleties lie in the form of the trajectory information that is being used.

In the tracker-dependent approach, the planner generates a smooth trajectory (only state sequence information is needed) in the configuration space, and relies on a separate trajectory tracker[1] to generate the instantaneous setpoint for an actuator to follow the trajectory while some other controller takes care of servoing the actuator. In practice, this architecture works well in many applications [Kuwata et al., 2008, Campbell et al., 2007, Wei et al., 2013], but the tracking control almost certainly undermines some

[1] An example of a trajectory tracker is the LQR-Controller used for smoothing discussed in Chapter 3. [Snider, 2009] gives a comprehensive review of popular trajectory trackers in the context of autonomous passenger vehicles.

intended behavior of the planner. For example, the pure pursuit tracker is commonly used for APV applications. But it exhibits corner-cutting behaviors for trajectories with high curvature [Snider, 2009], and rarely does the planner explicitly model the tracking controller[2].

The open-loop approach is different from the tracker-dependent approach: the trajectory (the control sequence and state sequence can be further obtained by evaluating the control over a selected vehicle model) generated by the planner will be directly used as the desired setpoint for the low-level actuator servo loop to follow. Obviously, this puts a more stringent requirement on the quality of the trajectory plan. The benefit is that the overall planning/control architecture is more straightforward with less delay, and requires the planner to be knowledgeable about the environment and the vehicle itself.

In reality, the trajectory tracker is imperfect, and it is difficult and cumbersome to characterize the tracking error within the planner. On the other hand, it is relatively easy to develop sufficiently complex models for on-road autonomous driving to describe the motion with good accuracy, such as the vehicle models explained in Section 3.1.1 and 3.1.2. As a result, we will take the open-loop approach for trajectory planning, and the choice of the subsequent trajectory optimizer must also be compatible with the form of trajectory needed for an open-loop planning approach, i.e., it must have an explicit control sequence. This is one of the primary reasons to choose the iLQR algorithm.

6.2 Trajectory Optimization Background

Section 2.2.3 reviewed both direct and indirect trajectory optimization methods. In this section, we summarize in detail one specific indirect (shooting-based) trajectory optimizer — the iterative linear quadratic regulator (iLQR). This optimizer will be used as the engine for local trajectory optimization.

[2]Since the initial objective of emplacing a tracking controller is to free the planner from worrying about the low-level dynamics in order to plan/control at a higher level.

The iterative-LQR (iLQR) is different from the LQR in that: in LQR, the optimal policy of a given state is calculated once assuming the future dynamics are precisely governed by the locally linearized system, while the iLQR recognizes that this assumption is not valid for a nonlinear system, and therefore iteratively performs LQR on the new state generated from the last optimization cycle, until a true local minimum is achieved.

The LQR controller was used in the trajectory tracking control for smoothing explained in Section 3.4. In this section, the goal of using the iLQR algorithm is no longer to minimize a state-related (tracking) quadratic error, but to minimize a more general (not necessarily quadratic, so it needs quadratization) cost function over an entire trajectory, in order to consider a broader range of factors that affect driving.

Now we can introduce the concept of trajectory: a state sequence ($N+1$ states) and a control sequence (N controls) over a fixed planning horizon T sampled at δt ($N \cdot \delta t = T$):

$$\begin{aligned} \boldsymbol{X}^{(k)} &\doteq \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_{N-1}, \boldsymbol{x}_N\} \\ \boldsymbol{U}^{(k)} &\doteq \{\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_i, \ldots, \boldsymbol{u}_{N-1}\} \end{aligned} \tag{6.1}$$

where $\boldsymbol{X}^{(k)}$ and $\boldsymbol{U}^{(k)}$ represent the state and control sequence of the k^{th}-iteration, and $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the state and control at the i^{th} time-stamp. A trajectory can be fully specified with an initial state $\boldsymbol{x}_0$, a control sequence $\boldsymbol{U}^{(k)}$ and the selected vehicle discrete dynamics model f_d, such that:

$$\boldsymbol{x}_{i+1} = f_d(\boldsymbol{x}_i, \boldsymbol{u}_i)$$

The optimization problem can be formulated as finding an optimal control sequence $\boldsymbol{U}^*$ that minimizes a total cost $J(\boldsymbol{X}, \boldsymbol{U})$ of the sum of intermediate costs g and the final cost g_N over the entire trajectory:

$$\boldsymbol{U}^* = \underset{\boldsymbol{U}}{\operatorname{argmin}}\, J(\boldsymbol{X}, \boldsymbol{U}) \tag{6.2}$$

where

$$J(\boldsymbol{X}, \boldsymbol{U}) = g_N(\boldsymbol{x}_N) + \sum_{i=0}^{N-1} g(\boldsymbol{x}_i, \boldsymbol{u}_i)$$

To represent the dynamic programming mechanism, given a control sequence, the value function (optimal cost-to-go) V of the i^{th} state is defined as the minimum sum of the intermediate cost g of the i^{th} state-control pair and the value function of the $(i+1)^{th}$ state,

$$V(\boldsymbol{x}_i) = \min_{\boldsymbol{u}_i}[g(\boldsymbol{x}_i, \boldsymbol{u}_i) + V(\boldsymbol{x}_{i+1})] \tag{6.3}$$

Performing dynamic programming on a DAG (Section 4.2) or a undirected graph (a.k.a, Dijkstra's algorithm [Hart et al., 1968]) is straightforward, since the states and transitions are discretized and finite. To perform dynamic programming on a system without discretized state/transition is tricky: a slight change $\delta\boldsymbol{u}$ (due to optimization) in a control $\boldsymbol{u}_i$ at the i^{th} time-stamp not only introduces change in the intermediate g, but also changes the $(i+1)^{th}$ state $\boldsymbol{x}_{i+1}$, whose value function is unknown and must be approximated. The core of the iLQR algorithm is a pair of forward and backward propagation routines that efficiently update the value functions while iteratively nudging the control sequence by a new $\delta\boldsymbol{u}$.

Let the function inside the min operator in Equation 6.3 be P:

$$\begin{aligned} P(\boldsymbol{x}_i, \boldsymbol{u}_i) &= g(\boldsymbol{x}_i, \boldsymbol{u}_i) + V(\boldsymbol{x}_{i+1}) \\ &= g(\boldsymbol{x}_i, \boldsymbol{u}_i) + V(f_d(\boldsymbol{x}_i, \boldsymbol{u}_i)) \end{aligned} \tag{6.4}$$

Suppose now that small variations in state and control $(\delta\boldsymbol{x}, \delta\boldsymbol{u})$ at $(\boldsymbol{x}_i, \boldsymbol{u}_i)$ are both incurred independently at the i^{th} time-stamp. $\delta\boldsymbol{x}$ is incurred by the state and control changes made to the optimization of the $(i-1)^{th}$ time-stamp, whereas $\delta\boldsymbol{u}$ is incurred by some control update (optimization) algorithm of the current i^{th} time-stamp. Define the change in P to be $Q(\delta\boldsymbol{x}, \delta\boldsymbol{u})$, i.e., the sum of the change in g and the change in value function V_{i+1} due to the change in the $(i+1)^{th}$ states,

$$\begin{aligned} Q(\delta\boldsymbol{x}, \delta\boldsymbol{u}) &= P(\boldsymbol{x}_i + \delta\boldsymbol{x}, \boldsymbol{u}_i + \delta\boldsymbol{u}) - P(\boldsymbol{x}_i, \boldsymbol{u}_i) \\ &= g(\boldsymbol{x}_i + \delta\boldsymbol{x}, \boldsymbol{u}_i + \delta\boldsymbol{u}) - g(\boldsymbol{x}_i, \boldsymbol{u}_i) \\ &\quad + V(f_d(\boldsymbol{x}_i + \delta\boldsymbol{x}, \boldsymbol{u}_i + \delta\boldsymbol{u})) - V(f_d(\boldsymbol{x}_i, \boldsymbol{u}_i)) \end{aligned} \tag{6.5}$$

Quadratizing Q around $(\mathbf{0}, \mathbf{0})$ as in Equation 3.12 with $\delta\boldsymbol{x} \to \mathbf{0}$ and $\delta\boldsymbol{u} \to \mathbf{0}$, and noting that Q is a function of $(\delta\boldsymbol{x}, \delta\boldsymbol{u})$ instead of $(\boldsymbol{x}_i, \boldsymbol{u}_i)$,

$$
\begin{aligned}
Q(\delta \boldsymbol{x}, \delta \boldsymbol{u}) \approx \tilde{Q}(\delta \boldsymbol{x}, \delta \boldsymbol{u}) &= Q(\mathbf{0}, \mathbf{0}) \\
&+ \frac{\partial Q(\delta \boldsymbol{x}, \delta \boldsymbol{u})}{\partial [\delta \boldsymbol{x}]}\Big|_{\mathbf{0},\mathbf{0}} [\delta \boldsymbol{x}] + \frac{\partial Q(\delta \boldsymbol{x}, \delta \boldsymbol{u})}{\partial [\delta \boldsymbol{u}]}\Big|_{\mathbf{0},\mathbf{0}} [\delta \boldsymbol{u}] + \frac{1}{2} [\delta \boldsymbol{x}]^T \frac{\partial^2 Q(\delta \boldsymbol{x}, \delta \boldsymbol{u})}{\partial [\delta \boldsymbol{x}]^2}\Big|_{\mathbf{0},\mathbf{0}} [\delta \boldsymbol{x}] \\
&+ [\delta \boldsymbol{u}]^T \frac{\partial^2 Q(\delta \boldsymbol{x}, \delta \boldsymbol{u})}{\partial [\delta \boldsymbol{u}] \partial [\delta \boldsymbol{x}]}\Big|_{\mathbf{0},\mathbf{0}} [\delta \boldsymbol{x}] + \frac{1}{2} [\delta \boldsymbol{u}]^T \frac{\partial^2 Q(\delta \boldsymbol{x}, \delta \boldsymbol{u})}{\partial [\delta \boldsymbol{u}]^2}\Big|_{\mathbf{0},\mathbf{0}} [\delta \boldsymbol{u}] \\
&= 0 + Q_x [\delta \boldsymbol{x}] + Q_u [\delta \boldsymbol{u}] + \frac{1}{2} [\delta \boldsymbol{x}]^T Q_{xx} [\delta \boldsymbol{x}] + [\delta \boldsymbol{u}]^T Q_{ux} [\delta \boldsymbol{x}] + \frac{1}{2} [\delta \boldsymbol{u}]^T Q_{uu} [\delta \boldsymbol{u}] \\
&= \frac{1}{2} \begin{bmatrix} 1 \\ \delta \boldsymbol{x} \\ \delta \boldsymbol{u} \end{bmatrix}^T \begin{bmatrix} 0 & Q_x^T & Q_u^T \\ Q_x & Q_{xx} & Q_{xu} \\ Q_u & Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta \boldsymbol{x} \\ \delta \boldsymbol{u} \end{bmatrix}
\end{aligned}
\tag{6.6}
$$

where

$$
\begin{aligned}
Q_x &= \frac{\partial [Q(\delta \boldsymbol{x}, \delta \boldsymbol{u})]}{\partial [\delta \boldsymbol{x}]}\Big|_{\mathbf{0},\mathbf{0}} = \frac{\partial g(\boldsymbol{x}_i + \delta \boldsymbol{x}, \boldsymbol{u}_i + \delta \boldsymbol{u})}{\partial [\delta \boldsymbol{x}]}\Big|_{\mathbf{0},\mathbf{0}} + \frac{\partial V(f_d(\boldsymbol{x}_i + \delta \boldsymbol{x}, \boldsymbol{u}_i + \delta \boldsymbol{u}))}{\partial [\delta \boldsymbol{x}]}\Big|_{\mathbf{0},\mathbf{0}} \\
&= \frac{\partial g}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i} + \frac{\partial f_d}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i}^T \cdot \frac{\partial V}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_{i+1}, \boldsymbol{u}_{i+1}} = g_x + f_x^T V_x' \\
Q_u &= \frac{\partial [Q(\delta \boldsymbol{x}, \delta \boldsymbol{u})]}{\partial [\delta \boldsymbol{u}]}\Big|_{\mathbf{0},\mathbf{0}} = \frac{\partial g(\boldsymbol{x}_i + \delta \boldsymbol{x}, \boldsymbol{u}_i + \delta \boldsymbol{u})}{\partial [\delta \boldsymbol{u}]}\Big|_{\mathbf{0},\mathbf{0}} + \frac{\partial V(f_d(\boldsymbol{x}_i + \delta \boldsymbol{x}, \boldsymbol{u}_i + \delta \boldsymbol{u}))}{\partial [\delta \boldsymbol{u}]}\Big|_{\mathbf{0},\mathbf{0}} \\
&= \frac{\partial g}{\partial \boldsymbol{u}}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i} + \frac{\partial f_d}{\partial \boldsymbol{u}}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i}^T \cdot \frac{\partial V}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_{i+1}, \boldsymbol{u}_{i+1}} = g_u + f_u^T V_x' \\
Q_{xx} &= \frac{\partial^2 [Q(\delta \boldsymbol{x}, \delta \boldsymbol{u})]}{\partial [\delta \boldsymbol{x}]^2}\Big|_{\mathbf{0},\mathbf{0}} = \frac{\partial^2 g(\boldsymbol{x}_i + \delta \boldsymbol{x}, \boldsymbol{u}_i + \delta \boldsymbol{u})}{\partial [\delta \boldsymbol{x}]^2}\Big|_{\mathbf{0},\mathbf{0}} + \frac{\partial^2 V(f_d(\boldsymbol{x}_i + \delta \boldsymbol{x}, \boldsymbol{u}_i + \delta \boldsymbol{u}))}{\partial [\delta \boldsymbol{x}]^2}\Big|_{\mathbf{0},\mathbf{0}} \\
&= \frac{\partial^2 g}{\partial \boldsymbol{x}^2}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i} + \frac{\partial f_d}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i}^T \cdot \frac{\partial^2 V}{\partial \boldsymbol{x}^2}\Big|_{\boldsymbol{x}_{i+1}, \boldsymbol{u}_{i+1}} \cdot \frac{\partial f_d}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i} + \frac{\partial V}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_{i+1}, \boldsymbol{u}_{i+1}} \cdot \frac{\partial^2 f_d}{\partial \boldsymbol{x}^2}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i} \\
&= g_{xx} + f_x^T \cdot V_{xx}' \cdot f_x + V_x' \cdot f_{xx} \\
Q_{ux} &= \frac{\partial^2 [Q(\delta \boldsymbol{x}, \delta \boldsymbol{u})]}{\partial [\delta \boldsymbol{u}] \partial [\delta \boldsymbol{x}]}\Big|_{\mathbf{0},\mathbf{0}} = \frac{\partial^2 g(\boldsymbol{x}_i + \delta \boldsymbol{x}, \boldsymbol{u}_i + \delta \boldsymbol{u})}{\partial [\delta \boldsymbol{u}] \partial [\delta \boldsymbol{x}]}\Big|_{\mathbf{0},\mathbf{0}} + \frac{\partial^2 V(f_d(\boldsymbol{x}_i + \delta \boldsymbol{x}, \boldsymbol{u}_i + \delta \boldsymbol{u}))}{\partial [\delta \boldsymbol{x}] \partial [\delta \boldsymbol{u}]}\Big|_{\mathbf{0},\mathbf{0}} \\
&= \frac{\partial^2 g}{\partial \boldsymbol{u} \partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i} + \frac{\partial f_d}{\partial \boldsymbol{u}}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i}^T \cdot \frac{\partial^2 V}{\partial \boldsymbol{x}^2}\Big|_{\boldsymbol{x}_{i+1}, \boldsymbol{u}_{i+1}} \cdot \frac{\partial f_d}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i} + \frac{\partial V}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_{i+1}, \boldsymbol{u}_{i+1}} \cdot \frac{\partial^2 f_d}{\partial \boldsymbol{u} \partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i} \\
&= g_{ux} + f_u^T \cdot V_{xx}' \cdot f_x + V_x' \cdot f_{ux} \\
Q_{uu} &= \frac{\partial^2 [Q(\delta \boldsymbol{x}, \delta \boldsymbol{u})]}{\partial [\delta \boldsymbol{u}]^2}\Big|_{\mathbf{0},\mathbf{0}} = \frac{\partial^2 g(\boldsymbol{x}_i + \delta \boldsymbol{x}, \boldsymbol{u}_i + \delta \boldsymbol{u})}{\partial [\delta \boldsymbol{u}]^2}\Big|_{\mathbf{0},\mathbf{0}} + \frac{\partial^2 V(f_d(\boldsymbol{x}_i + \delta \boldsymbol{x}, \boldsymbol{u}_i + \delta \boldsymbol{u}))}{\partial [\delta \boldsymbol{u}]^2}\Big|_{\mathbf{0},\mathbf{0}} \\
&= \frac{\partial^2 g}{\partial \boldsymbol{u}^2}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i} + \frac{\partial f_d}{\partial \boldsymbol{u}}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i}^T \cdot \frac{\partial^2 V}{\partial \boldsymbol{x}^2}\Big|_{\boldsymbol{x}_{i+1}, \boldsymbol{u}_{i+1}} \cdot \frac{\partial f_d}{\partial \boldsymbol{u}}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i} + \frac{\partial V}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_{i+1}, \boldsymbol{u}_{i+1}} \cdot \frac{\partial^2 f_d}{\partial \boldsymbol{u}^2}\Big|_{\boldsymbol{x}_i, \boldsymbol{u}_i} \\
&= g_{uu} + f_u^T \cdot V_{xx}' \cdot f_u + V_x' \cdot f_{uu}
\end{aligned}
$$

Notice that each of the second-order terms Q_{xx}, Q_{ux} and Q_{uu} above has the second-order dynamics differentials f_{xx}, f_{ux} and f_{uu}. This is a generic differential dynamic programming formulation. For iLQR, it is a common practice to assume these second-order dynamics differentials to be zero.

We find the optimal control variation $\delta \boldsymbol{u}^*$ by minimizing the quadratized $\tilde{Q}$:

$$\delta \boldsymbol{u}^* = \underset{\delta u}{\operatorname{argmin}} \tilde{Q}(\delta \boldsymbol{x}, \delta \boldsymbol{u}) \tag{6.7}$$

A solution can be found analytically by solving,

$$\frac{\partial \tilde{Q}(\delta \boldsymbol{x}, \delta \boldsymbol{u})}{\partial [\delta \boldsymbol{u}]} = Q_{\boldsymbol{u}} + Q_{\boldsymbol{ux}} \cdot \delta \boldsymbol{x} + Q_{\boldsymbol{uu}} \cdot \delta \boldsymbol{u} = 0 \tag{6.8}$$

We can get

$$\begin{aligned} \delta \boldsymbol{u}^* &= -Q_{\boldsymbol{uu}}^{-1}(Q_{\boldsymbol{u}} + Q_{\boldsymbol{ux}} \cdot \delta \boldsymbol{x}) \\ &= \boldsymbol{k} + \boldsymbol{K} \cdot \delta \boldsymbol{x} \end{aligned} \tag{6.9}$$

where

$$\begin{aligned} \boldsymbol{k} &= -Q_{uu}^{-1} Q_u \\ \boldsymbol{K} &= -Q_{uu}^{-1} Q_{ux} \end{aligned} \tag{6.10}$$

It is worth mentioning again that the changes in state $\delta \boldsymbol{x}$ and control $\delta \boldsymbol{u}$ are not correlated. Before the calculation of the optimal update $\delta \boldsymbol{u}$ for the i^{th} control, the change in the i^{th} state $\delta \boldsymbol{x}$ is already calculated. In the calculation of the optimal control variation, we need to calculate $Q_{\boldsymbol{u}}$, $Q_{\boldsymbol{uu}}$, $Q_{\boldsymbol{ux}}$, which are related to the value functions $\frac{\partial V}{\partial \boldsymbol{x}}$ and $\frac{\partial^2 V}{\partial \boldsymbol{x}^2}$ of the next timestamp. Therefore, it is important to get an explicit representation of them by plugging the result of Equation 6.9 into the quadratized Equation 6.6, and taking the first- and second-order derivatives w.r.t. $\delta \boldsymbol{x}$, we can further get:

$$\begin{aligned} \frac{\partial V}{\partial \boldsymbol{x}} &= Q_x - Q_u Q_{uu}^{-1} Q_{ux} \\ \frac{\partial^2 V}{\partial \boldsymbol{x}^2} &= Q_{xx} - Q_{xu} Q_{uu}^{-1} Q_{ux} \end{aligned} \tag{6.11}$$

iLQR Algorithm: The standard iLQR algorithm can be summarized as follows:

1. Perform a forward shooting procedure from the initial state $\boldsymbol{x}_0$ with the initial control sequence $\boldsymbol{U}^{(0)}$ to get the initial state sequence $\boldsymbol{X}^{(0)}$.

2. In a loop indexed by k, starting from $k = 0$, calculate the control update gains $\boldsymbol{k}_i$ and $\boldsymbol{K}_i$ for each state/control pair $(\boldsymbol{x}_i, \boldsymbol{u}_i)$ in a reverse fashion with Equations 6.6 through 6.11.

3. Perform the optimization with the gains above in a forward shooting procedure, starting from $i = 0$ to obtain the new state/control sequence $\boldsymbol{X}^{(k+1)}/\boldsymbol{U}^{(k+1)}$.

   - Generate $\delta \boldsymbol{x}_i = \boldsymbol{x}_i^{(k+1)} - \boldsymbol{x}_i^{(k)}$.
   - Generate $\delta \boldsymbol{u}_i = \boldsymbol{k}_i + \boldsymbol{K}_i \cdot \delta \boldsymbol{x}_i$.
   - Update the control with $\boldsymbol{u}_i^{(k+1)} = \boldsymbol{u}_i^{(k)} + \delta \boldsymbol{u}_i$
   - Obtain the new state with $\boldsymbol{x}_{i+1}^{(k+1)} = f_d(\boldsymbol{x}_i^{(k+1)}, \boldsymbol{u}_i^{(k+1)})$

4. Evaluate the trajectory cost before and update the optimization process,$J^{(k)} = J(\boldsymbol{X}^{(k)}, \boldsymbol{U}^{(k)})$, $J^{(k+1)} = J(\boldsymbol{X}^{(k+1)}, \boldsymbol{U}^{(k+1)})$.

5. The algorithm is terminated upon convergence based on the change in the cost function:

$$|\frac{J^{(k+1)} - J^{(k)}}{J^{(k)}}| < \epsilon_J \tag{6.12}$$

   where ϵ_J is the convergence threshold.

Note that the crux in the control update process is the update of $\boldsymbol{k}$ and $\boldsymbol{K}$, in which the inverse of Q_{xx} must be computed. Given a state/value pair, if the cost function nearby cannot be quadratized nicely, Q_{xx} might become numerically degenerate. In such a situation, the use of singular value decomposition for inverse calculation, the regulation of the inverse to make it positive definite, and the use of the Levenberg-Marquardt heuristic can help with a robust solver [Li and Todorov, 2004a]. The key goal now is to develop a reasonably shaped cost function and to start with a reasonable initial trajectory (using techniques from Chapter 3, 4 & 5).

6.3 Cost Function Design

We have shown that the standard iLQR solver is efficient with a simple analytical solution in the control variation, which presents the overall optimization process as an unconstrained optimization problem. The challenges now are two-fold:

1. Different (feature) cost terms must be crafted to be optimizer-friendly and to shape the APV's behavior as desired.

2. Different constraints must also be imposed, e.g., states must be restricted, the control has upper limits, etc.

In this section, we first review and draw intuition from the designed cost functions (terms) in prior work and summarize the key requirements in the design process for trajectory planning. In preparation of the cost function design, we describe the related theory in convex optimization and the selection of modulation. Finally, we explain our design of the behavioral/constraint feature functions to construct cost terms for the on-road driving application, and explore the convexity of these terms and their implications.

6.3.1 Cost function design overview

The iLQR algorithm and the planning methods reviewed in Section 2.2, whether discrete (control / sampling / search-based) or continuous (optimization-based), all depend on a predetermined optimality criterion to proceed, which ultimately shapes the planning outcome. Without loss of generality, we use the cost function (as opposed to a reward function) in the context of this thesis.

The classic path planning problem finds the minimum-length collision-free path, where the cost function consists solely of a non-negative cost term related to the length of the planned path. In APV self-driving, however, many aspects other than path length must be taken into account, and the most convenient formulation is a single-objective-multiple-feature (SOMWF) cost function. For a state/control pair $(\boldsymbol{x}_i, \boldsymbol{u}_i)$

along the trajectory, we define g/g_N as the intermediate/final cost functions which are calculated as the sum of M features:

$$\begin{aligned} g(\boldsymbol{x}_i, \boldsymbol{u}_i) &= \sum_{k=1}^{M} \omega_k \cdot c_k(\boldsymbol{x}_i, \boldsymbol{u}_i) \\ g_N(\boldsymbol{x}_N) &= \sum_{k=1}^{M} \omega_k \cdot c_k(\boldsymbol{x}_N, \mathbf{0}) \end{aligned}$$

where ω_k are the weights of the corresponding cost terms c_k, and the final state $\boldsymbol{x}_N$ is the goal, so no control is applied to the final state. The total cost is obtained by accumulating the N costs associated with each state/control pair along the trajectory:

$$J(X, U) = g_N(\boldsymbol{x}_N) + \sum_{i=1}^{N-1} g(\boldsymbol{x}_i, \boldsymbol{u}_i) \tag{6.13}$$

Note that this cost representation is quite generic. It can represent not only the cost over the typical discretized trajectory sequence, but also can be adapted to represent the cost of a graph plan. Typically, on a graph, one needs to evaluate (and assign costs to) edges (state transitions) rather than nodes (states). The trick is to regard the cost associated with the edge as the cost incurred by one state and a control, which causes the state transition.

Cost terms associated with trajectory curvature, deviation from the speed limit, lateral accelerations and the distance to various environmental elements, such as surrounding obstacles, lane boundaries, centerlines, etc., are commonly designed for autonomous driving robots [Urmson et al., 2008, McNaughton, 2011, Ziegler et al., 2014]. We draw inspiration from this prior work using different planning techniques, and propose the following general guidelines in the cost function design process:

- *the need for* ***undesirable features***: a feature represents a certain undesirable aspect of a trajectory, which will incur a cost term. By choosing to design cost terms (as opposed to the reward terms), we can always rephrase the desirable features, such as ensuring that the car moves down the road, as undesirable features, e.g., to not move down the road.

- *the need for* ***cost shaping / modulation functions***: depending on how severely one wants to penalize a (undesirable) feature, one can choose a different shaping/modulation function to generate the cost term.
- *the preferability of* ***cost convexity / differentiability****:* for any graph-based planner, this is not a strong requirement since graph search is global and depends only on the immediate cost, rather than its gradients. However, for an optimization-based planner, if the cost function is non-convex, it will create different local minima in the search space. If non-convexity is unavoidable, it is crucial to design fewer non-convex terms and use global methods (e.g., the methods of Chapter 4 and 5) to compensate. In the meantime, the analytical gradient information is important for the efficiency of the optimization process.

6.3.2 Fundamental Result in Convex Optimization

Several fundamental results in convex analysis theory will be reviewed to provide guidelines for cost term design.

Convexity: a real-valued function $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is convex if

$$f(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leq \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y}), \forall \boldsymbol{x}, \boldsymbol{y} \in dom(f), \theta \in [0,1]$$

First-order Condition: a real-valued first-order differentiable function $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T (\boldsymbol{y} - \boldsymbol{x}), \forall \boldsymbol{x}, \boldsymbol{y} \in dom(f)$$

Second-order Condition: a real-valued second-order differentiable function $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if

$$\nabla^2 f(\boldsymbol{x}) \succeq 0, \forall \boldsymbol{x} \in dom(f)$$

In many situations, we need to apply some function to a feature/constraint to yield the cost term, which leads to function composition. The following lemma holds:

Lemma 6.3.1 *Let $h(y)$ be a second-order differentiable monotonically non-decreasing convex function $\mathbb{R} \to \mathbb{R}$, and $g(\boldsymbol{x})$ be a second-order differentiable convex function $\mathbb{R}^n \to \mathbb{R}$. Then the composition $(h \circ g)(\boldsymbol{x})$ is still convex. Note that monotonicity is only defined for $dom(h) \in \mathbb{R}$.*

Proof:

$$f = (h \circ g)(\boldsymbol{x}) = h(g(\boldsymbol{x}))$$

Taking the Hessian of f, because h is non-decreasing convex, we have $h_{\boldsymbol{g}}(g) \geq 0, h_{\boldsymbol{gg}}(g) \geq 0$, and because g is convex, we have $g_{\boldsymbol{xx}}(\boldsymbol{x}) \succeq 0$:

$$H_f = f_{\boldsymbol{xx}} = h_{\boldsymbol{gg}}(g) \cdot g_{\boldsymbol{x}}(\boldsymbol{x}) \cdot g_{\boldsymbol{x}}^T(\boldsymbol{x}) + h_{\boldsymbol{g}}(g) \cdot g_{\boldsymbol{xx}}(\boldsymbol{x}) \succeq 0$$

The composition function f is therefore convex, ■.

Oftentimes, we need to define the overall cost function by combining multiple cost terms, e.g., the sum of multiple weighted cost terms. For this purpose, we have the following lemma:

Lemma 6.3.2 *If two real-valued second-order differentiable functions $g(\boldsymbol{x})$ and $h(\boldsymbol{y})$ are convex, then a conical combination function f is still convex. $\boldsymbol{x}$ and $\boldsymbol{y}$ may or may not be the same variable/set.*

Proof: If $\boldsymbol{x} = \boldsymbol{y}$, we have:

$$f(\boldsymbol{x}) = \alpha \cdot g(\boldsymbol{x}) + \beta \cdot h(\boldsymbol{x})$$

If we take the second-order derivatives on both sides w.r.t. $\boldsymbol{x}$ to get the Hessian, and since we know g and h are convex, $H_g \succeq 0$, $H_h \succeq 0$, then a linear combination of two positive semi-definite matrices is also semi-definite:

$$H_f(\boldsymbol{x}) = \alpha \cdot H_g(\boldsymbol{x}) + \beta \cdot H_h(\boldsymbol{x}) \succeq 0$$

where H_f, H_g, H_h are the Hessians of functions f, g, h.

If $\boldsymbol{x} \neq \boldsymbol{y}$, we have

$$f(\boldsymbol{x}, \boldsymbol{y}) = \alpha \cdot g(\boldsymbol{x}) + \beta \cdot h(\boldsymbol{y}), \forall \alpha, \beta \geq 0$$

Taking the Hessian of f, we have:

$$H_f = \begin{bmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy} \end{bmatrix} = \begin{bmatrix} \alpha \cdot g_{xx} & \mathbf{0} \\ \mathbf{0} & \beta \cdot h_{yy} \end{bmatrix} = \begin{bmatrix} \alpha \cdot H_g & \mathbf{0} \\ \mathbf{0} & \beta \cdot H_h \end{bmatrix} \succeq 0$$

Regardless of the domains of g and h, their conical combination f is convex, ■.

6.3.3 Distance Functions

Features are the quantities that measure certain properties of the state and control. Sometimes, the state/control themselves are useful features (e.g, speed, swirl, acceleration, etc.). Some states, like absolute position, are neutral quantities, but relative position (distances) can be extracted as useful features.

The calculation of distances is an important step in feature definition. On a discrete canvas (e.g., a grid map), the distance transform [Breu et al., 1995, Felzenszwalb and Huttenlocher, 2004], Voronoi diagram [Aurenhammer, 1991, Dolgov et al., 2008], and shape-convolution algorithms [Kavraki, 1995] have been developed to extract the distance information. Straightforward distance computation between polygonal shapes can be achieved with polygon-distance algorithms like [Gilbert et al., 1988, den Bergen, 1999], which are comparatively expensive. If the shape of the obstacles can be approximated by circular disks [Melissen and Schuur, 2000, Ziegler and Stiller, 2010], we can straightforwardly use the Euclidean norm for distance computation. If distances to a polyline are needed, the naive approach is to calculate the distance to each line segment and take the minimum. A more efficient and 2^{nd}-order continuous method makes use of the so-called "signed lp-distance" to calculate a smooth distance function analytically [Belyaev et al., 2013]. Following the cost function requirements stated in Section 6.3.1, only second-order continuous and convex distance functions, Euclidean norm and signed lp-distance, are used.

Given a query point $\boldsymbol{x}$ and a target point $\boldsymbol{x}_0$, the Euclidean norm (2-norm) gives a natural distance measurement:

Euclidean Norm Distance:

$$d_{norm}(\boldsymbol{x}|\boldsymbol{x}_0) = \|\boldsymbol{x} - \boldsymbol{x}_0\|_2$$

Lemma 6.3.3 *The norm function is convex.*

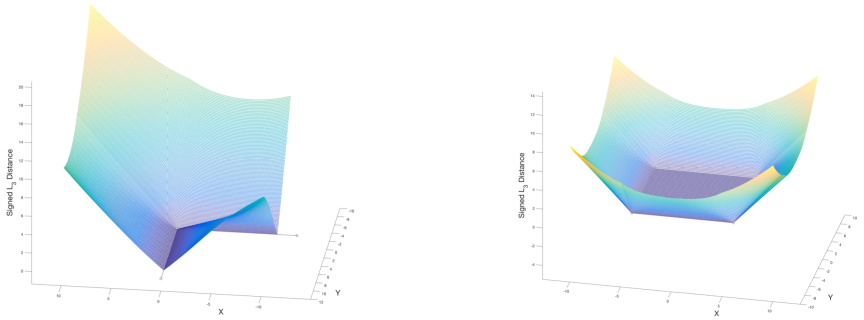
Proof: By the triangle inequality,

$$d_{norm}(\lambda \cdot \boldsymbol{x} + (1-\lambda) \cdot \boldsymbol{y}) \leq d_{norm}(\lambda \cdot \boldsymbol{x}) + d_{norm}((1-\lambda) \cdot \boldsymbol{y})$$

By homogeneity,

$$d_{norm}(\lambda \cdot \boldsymbol{x} + (1-\lambda) \cdot \boldsymbol{y}) \leq \lambda d_{norm}(\boldsymbol{x}) + (1-\lambda) d_{norm}(\boldsymbol{y}), \blacksquare.$$

Often times, it is necessary to calculate the distance of a given point to a polyline or a polygon. Traditional methods are often inefficient and yields noncontinuous solution with an algorithmic approach. The signed lp-distance [Belyaev et al., 2013] (Figure 6.1) provides a physics-inspired smooth analytical approach:



(a) Distance to polyline (b) Distance to polygon

Figure 6.1: Distance functions for polyline and polygon.

Signed L_p Distance: given a polyline defined by $\mathcal{L} = \{l_0, l_1, \ldots, l_{N-1}\}$, the signed L_p-distance is given by:

$$d_{lp}(\boldsymbol{x}|\mathcal{L}) = [\frac{1}{\varphi_p(\boldsymbol{x})}]^{\frac{1}{p}} \tag{6.14}$$

where $\varphi_p(\boldsymbol{x})$ is the so-called double-layer potential function on a smooth oriented hyper-surface $S \in \mathbb{R}^2$ (we are concerned only with the 2-D case):

$$\varphi_p(\boldsymbol{x}) = \int_\Omega \frac{d\Omega_y}{|\boldsymbol{x} - \boldsymbol{y}|^p} \tag{6.15}$$

where $y \in S$, $d\Omega_y$ is some solid angle at which the surface element dS_y is seen from $\boldsymbol{x}$. With a choice of an odd p, $\varphi_p(\boldsymbol{x})$ has an analytical form, e.g.,

$$\varphi_3(\boldsymbol{x}) = \sum_{i=0}^{N-1} \frac{t_i}{3}[\frac{1}{\boldsymbol{a}_i^3} + \frac{1}{\boldsymbol{b}_i^3}] + \frac{t_i + t_i^3}{6}[\frac{1}{\boldsymbol{a}_i} + \frac{1}{\boldsymbol{b}_i}]^3$$

while for each line segment, the start and end points of l_i are represented by $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$, $t_i = tan(\gamma_i/2)$, and γ_i is the angle between $\boldsymbol{x}\boldsymbol{a}_i$ and $\boldsymbol{x}\boldsymbol{b}_i$.

Lemma 6.3.4 *Signed l_p distance function is convex.*

Proof: In Equation 6.15, $|\boldsymbol{x} - \boldsymbol{y}|^p$ is convex, so $\frac{1}{|\boldsymbol{x}-\boldsymbol{y}|^p}$ is concave, so the integral is also concave. Taking the inverse again in Equation 6.14 makes the final function convex, ■.

6.3.4 One-sided Modulation Functions

To facilitate the calculation of cost terms, a few scalar modulation functions $f : \mathbb{R} \to \mathbb{R}$ are listed in Table 6.1. Note that the form of each function is taken as a one-sided version: use the corresponding form when $x \geq a$, use 0 when $x < a$. It can be shown [Bauschke et al., 2014] that all these functions are monotonically non-decreasing and convex. Note that the negentropy function is a preferred alternative to the affine function with similar shape, but second-order differentiability.

The combined use of these modulation functions can demonstrate great versatility in shaping the entire cost manifold of a single feature/constraint x. In other words, given a particular argument x, we can easily break it into two parts $x \geq x_r$ and $x < x_l$, and define the following combined convex function:

$$f(x) = f_1(x|a = x_r) + f_2(-x|a = x_l)$$

where $f_1|a = x_r$ and $f_2|a = x_l$ are any convex function listed in Table 6.1.

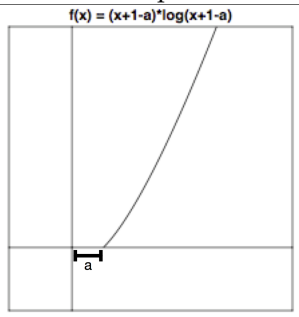
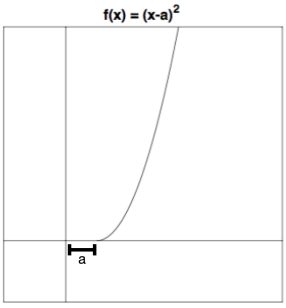
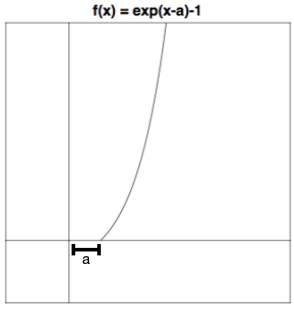
Name	Function	Shape
Negentropy	$f_{nentropy}(x\|a) = \begin{cases} y \cdot log(y), & \text{if } x \geq a \\ 0, & \text{otherwise} \end{cases}$ $y = x + 1 - a$	
Quadratic	$f_{quad}(x\|a) = \begin{cases} (x-a)^2, & \text{if } x \geq a \\ 0, & \text{otherwise} \end{cases}$	
Exponential	$f_{exp}(x\|a) = \begin{cases} e^{x-a} - 1, & \text{if } x \geq a \\ 0, & \text{otherwise} \end{cases}$	

Table 6.1: Examples of non-decreasing convex modulation functions.

Proof: If $f_2(x|a = x_l)$ is convex, then by Lemma 6.3.1, $f_2(-x|a = x_l)$ is also convex. Then $f_1(x) + f_2(-x)$ is also convex by Lemma 6.3.2, ■.

6.3.5 Behavioral/Constraint Cost

A behavioral cost fine-tunes the trajectory to enhance its comfort, while a constraint cost heavily penalizes the constraint violation to bring the APV back to an execution-feasible state. For each state/control pair $(\boldsymbol{x}_i, \boldsymbol{u}_i)$ along a trajectory, the intermediate

cost $g(\boldsymbol{x}_i, \boldsymbol{u}_i)$ is calculated as the sum of multiple behavioral/constraint cost terms $c^B(\boldsymbol{x}_i, \boldsymbol{u}_i)$ / $c^C(\boldsymbol{x}_i, \boldsymbol{u}_i)$, while the final cost $g_N(\boldsymbol{x}_N)$ is calculated as the sum of multiple final behavioral cost terms and constraint cost terms $c^B(\boldsymbol{x}_N, \boldsymbol{0})$ / $c^C(\boldsymbol{x}_N, \boldsymbol{0})$.

All cost terms are defined as the composite of a modulation function (preferably f_{quad} since it is quadratic and has an easily calculated Jacobian and Hessian) and a feature function χ. Since the modulation functions are one-sided non-decreasing, the feature function design philosophy is simple:

1. The feature function should yield a scalar value that quantifies a certain trajectory undesirability.

2. If the feature function is convex, the corresponding cost term will also be convex, according to Lemma 6.3.1. Therefore choose as many convex feature functions as possible to make the overall planning problem more convex.

If all the cost terms are convex, by Lemma 6.3.2, $g(\boldsymbol{x}_i, \boldsymbol{u}_i)$, its value-function V (optimal cost-to-go in Equation 6.3), the entire cost of the trajectory and the Q function in Equation 6.5 are all convex. In practice, however, not all useful cost terms are convex, such as the obstacle feature calculated below.

It is worth mentioning that constraint cost terms are important in guaranteeing the execution feasibility of the final trajectory. The initial trajectory fed to the trajectory optimizer is the outcome after applying techniques in Chapter 3, and is guaranteed to be model-feasible. However, model feasibility is not equivalent to execution feasibility. The vehicle dynamics models presented in Chapter 3 are just differential equations, which do not have any constraints. In fact, we cannot constrain dynamics by clamping the state, which will cause model discontinuities that make the calculation of partial derivatives ($f_{\boldsymbol{x}}$ and $f_{\boldsymbol{u}}$ in Equation 6.6) problematic. There also exist constraints that cannot be directly clamped independently.

In the following, behavioral feature/cost functions will be represented as χ^B and c^B; constraint feature/cost functions will be represented as ξ^C and c^C.

Obstacle Costs associated with obstacles are important for avoidance/distance keeping maneuvers. The APV and objects are approximated by one or more circular disks using techniques described in Section 3.2. Non-separable objects, like the curb of the road, are represented by polylines.

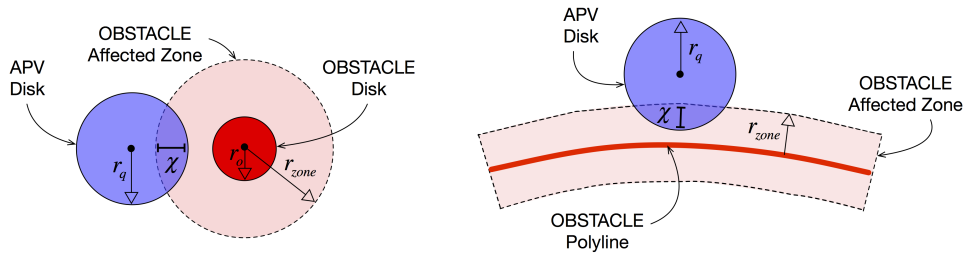


Figure 6.2: The calculation of penetrated distances with respect to disk and polyline.

Given a single query disk $(\boldsymbol{x}_q, r_q)$ and test against an obstacle in disk $(\boldsymbol{x}_o, r_o)$ form or a polyline $\mathcal{L}$ form, the distance between the two can be easily calculated with the Euclidean distance functions, but the feature we are interested in is the penetrated distance to the obstacle-affected zone incurred around the obstacle (Figure 6.2):

$$\begin{aligned} \chi^B_{obstacle} &= max(r_{zone} + r_q - d_{norm}(\boldsymbol{x}_q|\boldsymbol{x}_o), 0) \\ \chi^B_{obstacle} &= max(r_{zone} + r_q - d_{lp}(\boldsymbol{x}_q|\mathcal{L})|, 0) \end{aligned} \tag{6.16}$$

where r_{zone} is the radius of the affected zone.

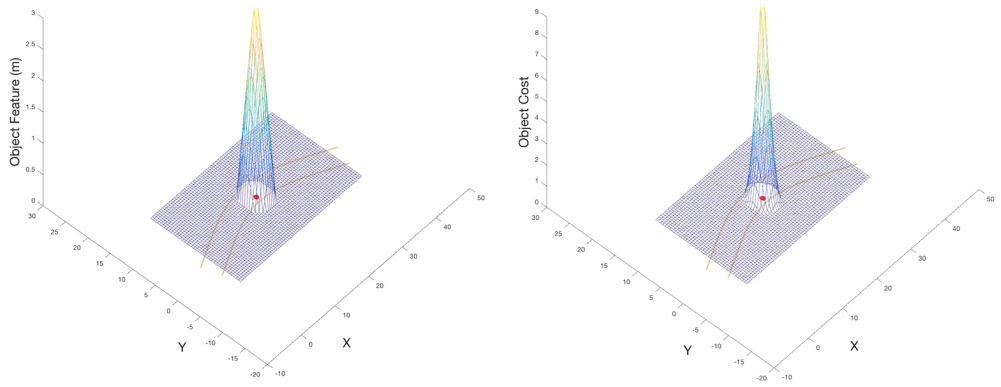
The feature for constraints is "the penetrated distance inside the obstacle itself":

$$\begin{aligned} \chi^C_{obstacle} &= max(r_q + r_o - d_{norm}(\boldsymbol{x}_q|\boldsymbol{x}_o), 0) \\ \chi^C_{obstacle} &= max(r_q - d_{lp}(\boldsymbol{x}_q|\mathcal{L}), 0), 0) \end{aligned} \tag{6.17}$$

The single-sided quadratic function to penalize penetration is further applied to calculate its cost term:

$$\begin{aligned} c^B_{obstacle} &= \omega_{obstacle} \cdot f_{quad}(\chi^B_{obstacle}) \\ c^C_{obstacle} &= \Omega \cdot f_{quad}(\chi^C_{obstacle}) \end{aligned} \tag{6.18}$$

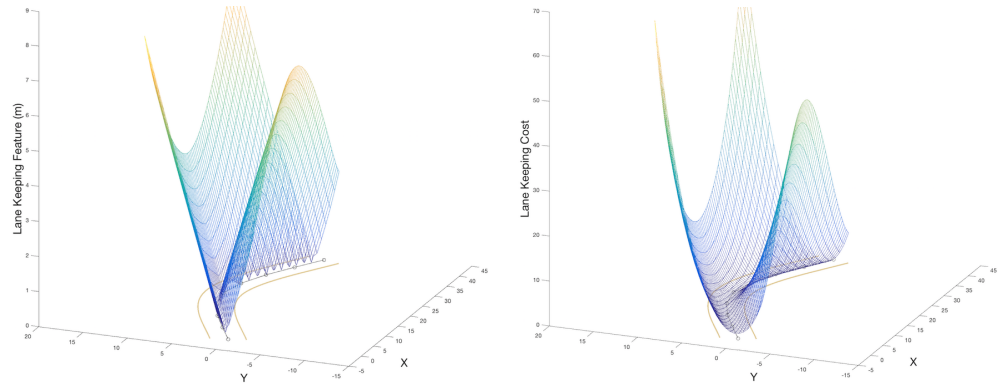
where $\omega_{obstacle}$ is the weight of the behavioral cost, and Ω is a large weight for all constraint cost terms.



(a) Behavioral feature (b) Cost

Figure 6.3: Feature & cost for object repulsion.

As shown in Figure 6.3, both feature/cost functions are non-convex, which makes the overall cost function non-convex. In fact, the obstacles create topological structure in the environment that leads to local minima. The optimizer will only converge to a local minimum from the initial trajectory based on the locally quadratized cost. This suggests the importance of starting off with an appropriate trajectory initialization through the techniques covered in Chapter 5.



(a) Behavioral feature (b) Cost

Figure 6.4: Feature & cost for lane keeping.

Reference Tracking Cost associated with the distance from the polyline of a reference path helps encourage the APV to follow the previously planned reference path (or simply following the lane center polyline). Given a query point x_q and a polyline $\mathcal{L}$ that represents the centerline, the lane-related behavioral feature is its distance to the polyline, while its constraint feature is its outer distance to the specified boundaries (typically the lane boundary):

$$\begin{aligned} \chi^B_{lane} &= d_{lp}(\boldsymbol{x}_q|\mathcal{L}) \\ \chi^C_{lane} &= max(d_{lp}(\boldsymbol{x}_q|\mathcal{L}) - d_{boundary}, 0) \end{aligned} \tag{6.19}$$

where $d_{boundary}$ is the boundary distance to the centerline polyline, e.g., half of the lane width.

Overall, both behavioral and constraint cost terms penalize greater deviation from the centerline:

$$\begin{aligned} c^B_{lane} &= \omega_{obstacle} \cdot f_{quad}(\chi^B_{lane}) \\ c^C_{lane} &= \Omega \cdot f_{quad}(\chi^C_{lane}) \end{aligned} \tag{6.20}$$

where $\omega_{obstacle}$ is the behavioral cost weight, and Ω is the constraint cost weight. Note that since both feature functions above are convex, the overall cost terms are also convex.

Speed In order to ensure the APV makes forward progress, a cost must be designed penalizing slow speed. Given the fixed planning horizon of the trajectory optimizer, there are two alternatives for this feature. One possibility is the spatial length of the trajectory. However, this is not a feature per state/control pair, but a single value over the entire trajectory. The more convenient behavioral feature is the difference between the current speed and the speed limit that is calculated per state, and a constraint feature is the speed difference that exceeds the allowed speed range:

$$\begin{aligned} \chi^B_{speed} &= max(v_{max} - v, 0) \\ \chi^C_{speed} &= max(max(v - v_{max}, 0), max(v_{min} - v, 0)) \end{aligned} \tag{6.21}$$

where v is the tangential speed of the vehicle, v_{max} is a non-negative speed limit, and v_{min} is the minimum speed, which is typically zero for on-road driving.

To penalize slow motion and heavily penalize states that are beyond the speed limit or below 0, we apply a quadratic function to this feature to get the cost term:

$$\begin{aligned} c^B_{speed} &= \omega_{speed} \cdot f_{quad}(\chi^B_{speed}) \\ c^C_{speed} &= \Omega \cdot f_{quad}(\chi^C_{speed}) \end{aligned} \tag{6.22}$$

where ω_{speed} is the behavioral cost weight, and Ω is the constraint cost weight. Note that the features are all convex, so both cost terms are also convex.

Lateral Acceleration High longitudinal speed on a curvy road is likely to introduce lateral acceleration. A behavioral feature will be the absolute value of the lateral acceleration itself, while the constraint feature will be the amount by which the allowed lateral acceleration is exceeded:

$$\begin{aligned} \chi^B_{latacc} &= |\kappa \cdot v^2| \\ \chi^C_{latacc} &= max(|\kappa \cdot v^2| - a^{latacc}_{max}, 0) \end{aligned} \tag{6.23}$$

where κ is the instantaneous curvature, v is the tangential speed, and a^{latacc}_{max} is the maximum lateral acceleration.

Excessive lateral acceleration is not only a potential source of discomfort, but also the source of undesirable slipping, which may cause safety concerns. To penalize excessive lateral acceleration, we develop the following cost terms:

$$\begin{aligned} c^B_{latacc} &= \omega_{latacc} \cdot f_{quad}(\chi^B_{latacc}) \\ c^C_{latacc} &= \Omega \cdot f_{quad}(\chi^B_{latacc}) \end{aligned} \tag{6.24}$$

where ω_{latacc} is the weight of the behavioral cost, and Ω is the constraint cost weight. Note that the features are all convex, so both cost terms are also convex.

Regardless of which vehicle model explained in Chapter 3 is used in the iLQR routine, the controls will have both swirl and longitudinal acceleration. The next two paragraphs design costs for both.

Swirl Control Swirl is the rate of steering. Similar behavioral and constraint features are designed:

$$\begin{aligned} \chi^B_{swirl} &= |\gamma| \\ \chi^C_{swirl} &= max(|\gamma| - \gamma_{max}, 0) \end{aligned} \tag{6.25}$$

where γ is the swirl command, and γ_{max} is the maximum absolute value of the swirl command.

High swirl causes aggressive steering change and motor weariness. Therefore, the following cost terms are used:

$$\begin{aligned} c^B_{swirl} &= \omega_{swirl} \cdot f_{quad}(\chi^B_{swirl}) \\ c^C_{swirl} &= \Omega \cdot f_{quad}(\chi^C_{swirl}) \end{aligned} \tag{6.26}$$

where ω_{swirl} is the behavioral weight for the swirl, and Ω is the constraining cost weight. Note that the features are all convex, so both cost terms are also convex.

Longitudinal Acceleration Control For longitudinal control, the behavioral feature is the signed value of the acceleration control itself, while the constraint feature is the acceleration difference that exceeds the allowed range:

$$\begin{aligned} \chi^B_{lonacc} &= a \\ \chi^C_{lonacc} &= max(max(a - a_{max}, 0), max(a_{min} - a)) \end{aligned} \tag{6.27}$$

where a is the longitudinal acceleration control, a_{max} is the maximum longitudinal acceleration, and a_{min} is the minimum longitudinal acceleration.

Excessive longitudinal (tangential) acceleration control should be penalized. Depending on whether the control is positive or negative, costs may be weighted differently to reflect the different tolerance levels of human passengers (people can tolerate much higher deceleration than acceleration).

$$\begin{aligned} c^B_{lonacc} &= \omega_{acc} \cdot f_{quad}(\chi^B_{lonacc}) + \omega_{dec} \cdot f_{quad}(-\chi^B_{lonacc}) \\ c^C_{lonacc} &= \Omega \cdot f_{quad}(\chi^C_{lonacc}) \end{aligned} \tag{6.28}$$

where ω_{acc} and ω_{dec} are respectively the weights for acceleration and deceleration, and Ω is the constraint cost weight. Note that by Lemma 6.3.2, the behavioral cost is still convex, so both cost terms are still convex.

6.4 Maneuver Pattern Constrained iLQR Algorithm

As discussed in the prior section, the existence of non-convex cost terms creates local minima in the solution space. An immediate implication is that the standard iLQR algorithm will only converge to a local minimum. Which local minimum it will converge to and the algorithm efficiency are highly dependent on the initialized trajectory. In Chapter 4 and 5, we have proposed a search-based global method accompanied by topological analysis tools to decide on the choice of maneuver pattern, which is a group of candidate trajectories. We want to choose an optimal trajectory from a particular group as the initial trajectory for the iLQR algorithm. This can be easily achieved by evaluating the candidate trajectories with the cost function designed in the prior section and picking the one with the minimum cost. However, adaptations to the standard iLQR algorithm are needed to improve optimization efficiency as well as to enforce a specific maneuver pattern constraint.

In the last paragraph of Section 6.2, we mentioned that in updating the control sequence, the computation of Q_{uu}^{-1} is required to calculate the control update gains $\boldsymbol{k}$ and $\boldsymbol{K}$. This is sometimes nontrivial for a poorly conditioned matrix. Regulation is needed by performing singular value decomposition (SVD) and adjusting its singular value matrix Σ:

$$Q_{uu} = P\Sigma Q^T \tag{6.29}$$

where the singular value matrix Σ is a non-negative real diagonal square matrix. In benign cases, the inverse of Q_{uu} can be calculated as:

$$Q_{uu}^{-1} = Q\Sigma^{-1}P^T \tag{6.30}$$

Notice that the matrix Σ^{-1} is obtained by taking the reciprocal of the diagonal ele-

ments (singular values) of Σ. In cases where some elements are zero (Σ is degenerate), the standard process [Li and Todorov, 2004b] is to add a small positive number λ to each of the singular values in Σ. λ serves as a regularization term to make sure the reciprocal of a singular value is never greater than $\frac{1}{\lambda}$. Notice that the greater λ is, the smaller $\boldsymbol{k}, \boldsymbol{K}$ are, according to Equation 6.10.

An important question is how λ should be determined, and how to enforce maneuver pattern constraints in the standard iLQR routine. The answer is that λ should be adjusted based on the result of the current optimization cycle. In each iLQR cycle, we define three possible outcome categories: poor, mediocre, and good. The naming of these categories is insignificant. What matters is the entry condition for each category, and how we should decide the result of the current optimization cycle and update λ for each. Once the new trajectory (state/control sequence) is updated with $\delta \boldsymbol{u} = \boldsymbol{k} + \boldsymbol{K} \cdot \delta \boldsymbol{x}$,

1. Re-examine if the new state sequence violates the determined maneuver pattern decided in Chapter 5. If a violation is detected, this will be called a "poor" cycle.

2. If the cost of the new trajectory is reduced, call it a "good" cycle; otherwise, $\delta \boldsymbol{u}$ does not have a good search direction or step size to reduce cost.

3. While it is nontrivial to figure out a better direction, it is relatively easy to adjust the step size by varying α in $\delta \boldsymbol{u} = \alpha \cdot \boldsymbol{k} + \boldsymbol{K} \cdot \delta \boldsymbol{x}$ as a practical method (the line-search method [Bertsekas, 1999]) to attempt to reduce cost. If line-search can improve the overall cost (if pattern constraint is violated, cost is infinity), call it a "mediocre" cycle.

4. At this point, neither the standard update nor the line-search can reduce the cost, call this a "poor" cycle.

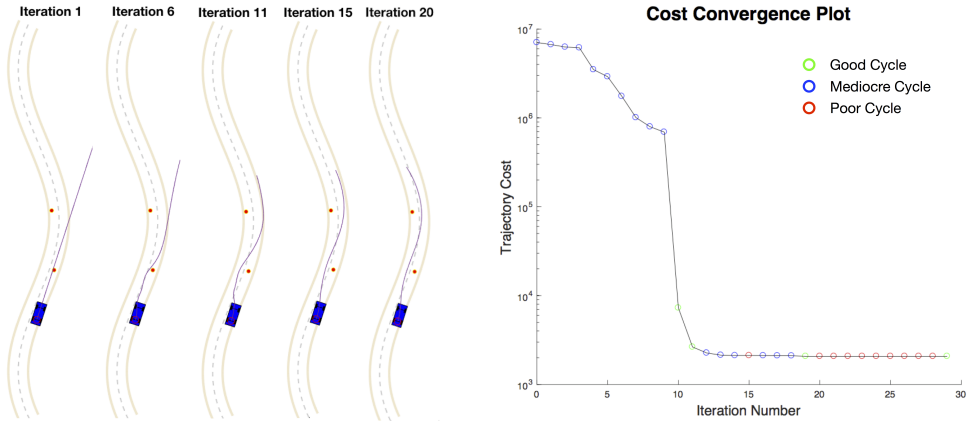
The three outcome categories indicate, with the current choice of λ, how good the iLQR-generated control update direction $\delta \boldsymbol{u}$ is w.r.t. reducing the overall trajectory cost. They are highly relevant to the quality of linearly approximated system dynamics

and quadratically approximated value function. Depending on the category, given a regulation scale factor $\beta_\lambda > 1$, we have three corresponding actions:

1. Good: we have reason to trust the iLQR direction in the next cycle to reduce the cost, so reduce $\lambda = \lambda / \beta_\lambda$, causing the optimizer to lean towards the second-order Newton method.

2. Mediocre: we are not sure if the iLQR search direction in the next cycle will reduce the cost, so maintain the trust level λ.

3. Poor: the current optimization either broke the maneuver pattern constraint or failed to reduce the cost even with line-search, which indicates that maybe the iLQR search direction is not trustworthy, so increase $\lambda = \lambda \cdot \beta_\lambda$, causing the optimizer to lean towards the first-order gradient descent method.



(a) World plot (b) Cost plot

Figure 6.5: Optimization demonstration of the adapted iterative-LQR trajectory optimizer on a poorly condition seeding trajectory violating both obstacle and lane boundary constraints.

Figure 6.5 demonstrates a challenging optimization scenario. The seeding trajectory is not well initialized since it violates multiple constraints (lane boundary and obstacle collision). However, it serves as a good example to demonstrate the robustness of the adapted iLQR algorithm. Empirically speaking, when constraints are violated,

the locally approximated LQR problem tends to be ill-conditioned, and the algorithm depends more on the line-search method to make steady progress in optimization. When the trajectory reaches a state in which the hard constraints are mostly solved (at iteration 10), leaning towards the standard iLQR control update (2^{nd} Newton Method) can reduce cost dramatically in a small number of iterations (iteration 11 & 12). This gives further insight on the importance of a good seeding trajectory in terms of starting with fewer constraint violations.

Chapter 7

Application to On-Road Self-Driving

In this chapter, we combine individual planning algorithms explained in Chapter 3, 4, 5, 6 systematically into a trajectory planning framework for on-road autonomous driving. Section 7.1 first explains the hierarchical structure of the planning framework and the responsibilities of each planning module. Section 7.2 specifies the parameter settings for each planning module. Section 7.3 demonstrates the planning results in several challenging test scenarios. Section 7.4 summarizes this chapter by comparing the proposed planner with the state of the art.

7.1 Trajectory Planning Framework

The design of the planning framework is inspired by two key ideas:

1. Human drivers have both a strategic understanding of the environment by knowing "roughly" how to maneuver in the form of a high-level maneuver decision and the muscle-level driving control for smooth maneuvering.

2. The driving task can be decomposed into two planning sub-problems to reduce the search space and enhance tunability. Humans tend to first identify environment topological structure and avoid stationary obstacles before reasoning about more complex maneuvers regarding the moving obstacles.

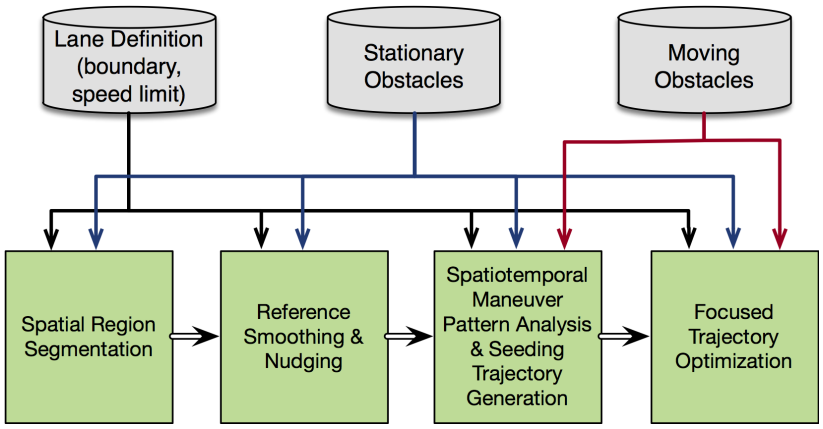


Figure 7.1: The planning algorithmic flow with localization/perception inputs.

With these ideas in mind, we propose a novel four-module trajectory planning framework (Figure 7.1):

1. M1—*Spatial Region Segmentation*: identify regions with different (topology-based) patterns considering the stationary obstacles.

2. M2—*Reference Smoothing & Nudging*: perform reference path smoothing and nudging avoidance with respect to the stationary obstacles subject to the selected spatial region segment constraints.

3. M3—*Spatiotemporal Maneuver Pattern Analysis & Seeding Trajectory Generation*: generate a pool of sample trajectories for two purposes. First, identify groups of trajectories with different (region-based, topology-based, sequence-based) patterns which further consider the moving obstacles; second, use the optimal trajectory in the selected group to seed the subsequent trajectory optimization process.

4. M4—*Trajectory optimization*: start with the seeding trajectory, and perform trajectory optimization subject to various constraints (including the maneuver pattern constraint).

The proposed planning framework echoes the first key inspiration idea in that both M1 and M3 provide maneuver pattern understanding at a tactical level, while M2 and

M4 perform the actual generation of the reference/trajectory. Regarding the second idea, the planning architecture generates a reference path plan first considering the stationary obstacles with modules M1 and M2. Then it generates the trajectory plan taking the moving obstacles into account.

7.2 Experiment Configuration

7.2.1 Robot vs. Simulation

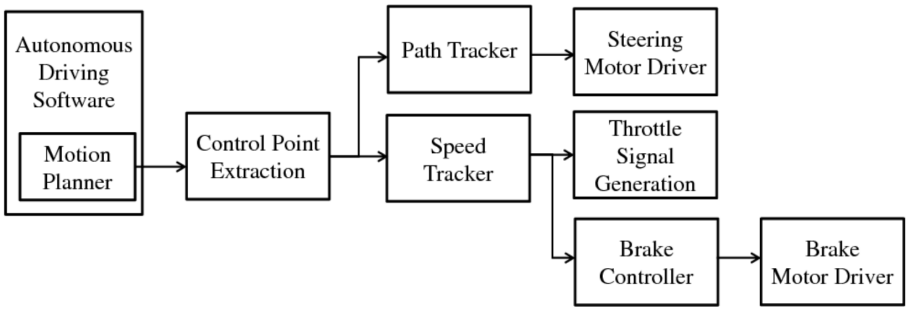


Figure 7.2: The planning/control diagram of the autonomous Cadillac SRX.

All planning components are evaluated in a lightweight simulation environment based on socket communication with Google's Protocol Buffer serialization. Partial evaluation was performed on the Cadillac SRX autonomous driving platform [Wei et al., 2013]. The planning algorithm runs on a Linux machine and communicates using the legacy software from the Tartan Racing Urban Challenge Operation System [Urmson et al., 2008]. It further sends the reference plan down to a drive-by-wire control system (illustrated in Figure 7.2) for path tracking control and handles moving obstacles with the simplified assumption of their being adaptive cruise control (ACC) targets. This is the classical tracker-dependent architecture discussed in Section 6.1.

The data interfaces between the planning modules and the robot/simulation environment are shown at a high level in Figure 7.3. The planning result of the reference smoothing and nudging can be interfaced directly with the Cadillac SRX for on-vehicle evaluation, while the overall trajectory result is evaluated in a simulation environment.

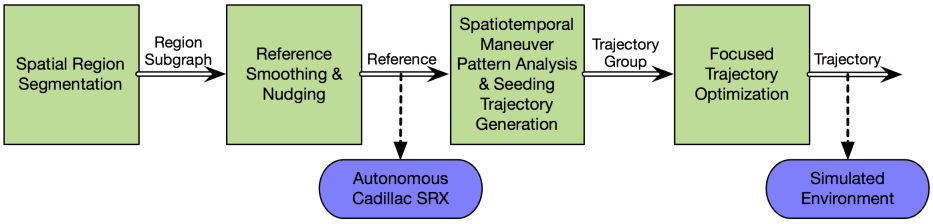


Figure 7.3: The outcomes of each proposed planning module in the algorithmic flow.

For both robot and simulation experiments, our algorithm is intended to run on any reasonably modern PC rather than exploiting the power of GPU processors, as in [McNaughton, 2011]. For example, the Cadillac SRX was equipped with four standalone PCs with Intel Core 2 Extreme Processor QX9300s, and the simulation environment has a CPU of similar computational power.

7.2.2 Spatial Region Segmentation

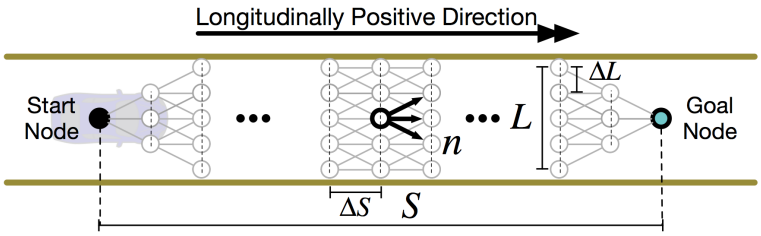


Figure 7.4: Graph specification for spatial region segmentation and reference smoothing/nudging.

The theory behind this module is explained in Section 5.2. To perform spatial region segmentation, the spatial nodes are first sampled conforming to the road structure in uniformly spaced longitudinal layers. Then directed edges are constructed only in the longitudinally positive direction between the neighboring layers, as shown in Figure 7.4. The segmentation result is represented as a sub-graph of this whole spatial graph. The five configuration parameters that determine the construction specification shown in Figure 7.4 are listed below:

- *Longitudinal look-ahead S*: 100 meters
- *Longitudinal resolution* ΔS: 4 meters
- *Lateral range L*: lane width.
- *Lateral resolution* ΔL: 0.4 meter
- *Node out-degree n*: the number of edges needed to fully connect to the nodes in the neighboring subsequent longitudinal layer.

The choice of these specific values is in general rule of thumb based on the our understanding of this application as well as trial and error process. But a few general principles apply. First, it is unwise to choose an overly long S, e.g., beyond the sensing horizon, since it increases the search space without actually providing the planner useful information to process. As for the ΔS and ΔL, they determine the sample granularity, which affects the ability of the constructed spatial graph to capture the topological structure of the continuous configuration space. Overall, the more densely the spatial graph is sampled, the more refined segmentation/planning capability this graph has, but at a cost of a greater computational overhead. Note The configuration of the spatial graph also affects the subsequent reference smoothing/nudging module.

7.2.3 Reference Smoothing & Nudging

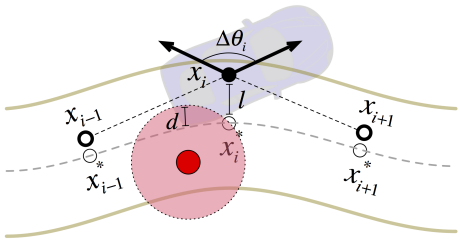


Figure 7.5: Cost specification for reference smoothing & nudging.

With the theory explained in Section 4.2, the edge-augmented graph is constructed based on the identified segmented spatial sub-graph from the prior module. Instead

of performing separate reference smoothing and nudging, we use a formulation that serves both purposes based on the same underlying edge-augmented graph construction technique. As shown in Figure 7.5, the cost function is defined very similar to Equation 4.1 with an additional obstacle-repulsive cost term c_{obs}:

$$
\begin{aligned}
c_{cl} &= \omega_{cl} \cdot \|\boldsymbol{x}_i - \boldsymbol{x}_i^*\|^2 \\
c_{sm} &= \omega_{sm} \cdot \frac{(\boldsymbol{x}_i - \boldsymbol{x}_{i-1})(\boldsymbol{x}_i - \boldsymbol{x}_{i+1})}{\|\boldsymbol{x}_i - \boldsymbol{x}_{i-1}\|\|\boldsymbol{x}_i - \boldsymbol{x}_{i+1}\|} \\
c_{obs} &= \omega_{obs} \cdot d
\end{aligned}
$$

where d is the penetrated distance inside the affected zone of an obstacle as explained Section 6.3.5. The weights of each cost term are the configurable parameters:

- *Centerline deviation weight ω_{cl}: 1.0*
- *Smoothness weight ω_{sm}: 15.0*
- *Obstacle repulsion weight ω_{obs}: 2.0*

Choosing these weighting parameters is also a rule of thumb practice. We take an empirical two-step process. Starting with a fixed non-negative ω_{cl}, e.g, 1.0, we tune ω_{sm} in order to achieve the desired behavior on high-curvature roads. Then, we tune for ω_{obs} in order to achieve the desired obstacle avoidance behavior. The final outcome of the edge-augmented graph is a sequence of linear spatial edges that track the road centerline with improved smoothness while avoiding the stationary obstacles. The piecewise-linear path is further smoothed through a LQR controller as explained in Section 3.4, and appended with a speed profile as explained in Section 4.3.

For smoothing, two sets of Q/R parameter matrices for the lateral and longitudinal trackers in Equation 3.21 to trade off tracking and control efforts must be specified. For simplicity, identity matrices are used.

For speed profile generation, the constraint parameters in Equation 4.5, 4.6 and 4.9 must be specified. Applying these constraints updates the speed limit based on the generated reference path. The constraint values are thus chosen to reflect the absolute maneuverability limits the of the APV:

- *Speed limit v_{max}: depending on the speed limit of the road, in m/s.*
- *Maximum longitudinal acceleration a^{lon}_{max}: 4.0 m/s^2.*
- *Maximum longitudinal deceleration d^{lon}_{max}: 8.0 m/s^2.*
- *Maximum longitudinal jerk j^{lon}_{max}: 10.0 m/s^3*

7.2.4 Spatiotemporal Maneuver Pattern Analysis

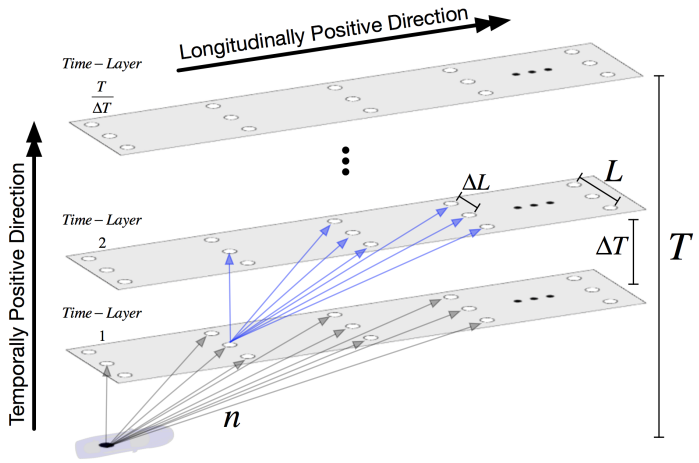


Figure 7.6: Graph specification for maneuver pattern analysis.

The theory behind this module is explained in Section 5.3. The candidate model-feasible trajectories are sampled in a two-step fashion. In the first step, the nodes are sampled per time-layer, which is, similar to Figure 7.4, a uniformly sampled grid along the determined reference. Then a graph of linear edges is constructed in the 3-D spatiotemporal domain, as in Figure 7.6.

This graph has the following features:

1. We do not sample covering the entire lane, but only sample around the reference obtained above.
2. Edges between spatial nodes from the same time-layer are not allowed, since no motion can happen without time elapsing.

3. Directed edges are constructed only when they move between neighboring time-layers in both temporally and longitudinally positive directions, since the APV is restricted to move forward in on-road driving.

4. If an edge has no longitudinal movement, it cannot have lateral movement, since the vehicle is non-holonomic.

In choosing the parameters, the parameter T determine the trajectory duration, which is wasteful computationally if chosen too long. On the other hand, if chosen to short, it is difficult for the planner to reason deliberatively. Once T is fixed, the parameter ΔT determines the number of time-layers. Since the out-degree of a node can be large, ΔT cannot be too small, which would introduce a large number of time-layers and would cause search space explosion. The parameters L and ΔL determine the number of lateral variations the planner will consider in spawning the possible trajectory candidates. But too refined granularity will greatly increase the out-degree number, and has negative impact on the overall computation overhead.

- *Trajectory time horizon* T: 6 seconds
- *Time resolution between time-layers* ΔT: 2 seconds
- *Lateral range* L: 1.0 meter
- *Lateral resolution* ΔL: 0.5 meter
- *Node out-degree* n: connect to all the longitudinally forward nodes such that the edge constructed has spatial length of s, and the average speed over this edge $\frac{s}{\Delta T}$ is no greater than the allowed maximum speed.

7.2.5 Trajectory Optimization

The theory behind this module is explained in Section 6.3. The temporal horizon T of a trajectory has been determined by the previous module, and we represent a trajectory as a T-second state/control discrete sequence sampled every δt seconds:

- *Trajectory time horizon* T: same as the last module.
- *Trajectory time resolution* δt: 0.1 second

Then according to Equation 6.1, there is a control sequence of length 60 that yields a state trajectory of length 61. The cost function g that shapes the eventual trajectory outcome have both behavioral and constraint-based cost terms as explained in Section 6.3.5 is given as:

$$\begin{aligned} g = & c^B_{obstacle}(\omega_{obstacle}) + c^B_{ref}(\omega_{ref}) + c^B_{speed}(\omega_{speed}) + c^B_{lat-acc}(\omega_{lat-acc}) + c^B_{swirl}(\omega_{swirl}) + c^B_{lon-acc}(\omega_{acc}, \omega_{dec}) \\ & + c^C_{obstacle}(\Omega) + c^C_{lane}(\Omega) + c^C_{speed}(\Omega) + c^C_{lat-acc}(\Omega) + c^C_{swirl}(\Omega) + c^C_{lon-acc}(\Omega) \end{aligned} \tag{7.1}$$

and the configurable weights of the corresponding cost terms are listed below:

- Weight for obstacle avoidance cost $\omega_{obstacle}$: 2.0
- Weight for reference tracking cost ω_{ref}: 3.0
- Weight for slow moving cost ω_{speed}: 1.0
- Weight for lateral acceleration cost ω_{latacc}: 1.0
- Weight for swirl control cost ω_{swirl}: 10.0
- Weight for longitudinal acceleration cost ω_{acc}: 1.0
- Weight for longitudinal deceleration cost ω_{dec}: 1.0
- Weight for constraint cost Ω: 30000

The continuous nature of these cost terms (w.r.t. the APV states) and the use of trajectory optimizer makes it significantly easier to observe the gradual changing of the trajectory after tuning a single weight compared to the sampling-based method [McNaughton, 2011]. However, the choice of the configurable weights is a still far from a trivial task, since the cost function conceptually should encode all the preferences in determining how the APV should behave in all situations and scenarios. In

practice, huge efforts on cost function tuning might be an inevitable engineering practice. Otherwise, one may resort to automated parameter tuning methods such as the maximum margin method [Ratliff et al., 2006] or the LEARCH algorithm [Silver et al., 2010]. The parameters listed above are specifically chosen based on the scenarios to be experimented in the following section.

7.3 Experimental Results

In this section, experiments are conducted to evaluate the proposed trajectory planning system. To better understand the results, the following three concepts are defined:

1. Scenario: a particular setup of the environment elements including lanes, obstacles and the APV.

2. Snapshot: plots of the environment elements' states and the planning outcomes of each planning module at a given time-stamp.

3. Overlay: a plot of the overlaid states of the environment elements' states over a time period to demonstrate the maneuver sequence.

For maybe 99% of the miles traveled, on-road driving is a boring task: one can simply follow the lane centerline while treating any possible interfering obstacles (resulting in head collision) as an adaptive cruise control (ACC) object. In order to highlight the features of the proposed algorithm, we designed some challenging on-road driving scenarios. Section 7.3.1 presents scenario I to demonstrate the navigation capability along an extremely curvy road segment. Section 7.3.2 presents scenario II to demonstrate topologically reasoning and swerving ability with respect to the stationary obstacles. Section 7.3.3 presents scenario III to demonstrate maneuver pattern reasoning and swerving ability with respect to the moving obstacles. Section 7.3.4 presents scenario IV to demonstrate obstacle avoidance ability while performing vehicle following.

In the following result presentation section, the snapshot plot is used to highlight the outcome and functionality of each of the planning modules of any fixed time-stamp. The overlay plot presents the full scenario handling result over a period of time. Five components are presented for each scenario:

1. **Spatial Region Segmentation**: snapshot plots showing the sampled global spatial on-road lattice graph and its identified sub-graphs representing topologically different patterns.

2. **Reference Smoothing & Nudging**: a snapshot plot of the reference plan generated by searching over the edge-augmented graph constructed from the selected lattice sub-graph. The reference is then used to guide the sampling of the seeding trajectories, and "attracts" the final optimized trajectory.

3. **Spatiotemporal Maneuver Pattern Analysis**: snapshot plots showing locally sampled and evaluated model-feasible trajectory candidates to explore distinct maneuver patterns and the selected seeding trajectory for the subsequent optimization process.

4. **Trajectory Optimization**: a snapshot plot of the execution-feasible optimized trajectory generated by further improving the seeding trajectory via a focused trajectory optimization module taking various constraints into account.

5. **Full Scenario Handling**: an overlay plot showing the APV's behavior for the next few seconds to demonstrate the full scenario handling sequence, and a plot of steering and speed history.

7.3.1 Scenario I: Curvy Road

In the scenario shown in Figure 7.7, we demonstrate the in-lane cruising capability on a curvy road. The goal is to traverse this high-curvature lane segment with reduced steering effort.

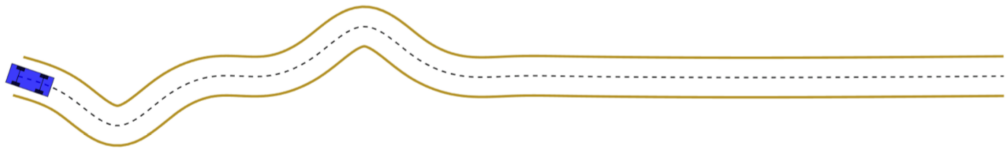


Figure 7.7: Scenario I

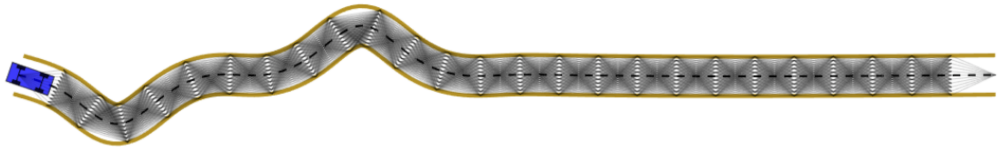


Figure 7.8: Segmented spatial region plot for scenario I.

Spatial Region Segmentation Since no stationary obstacles exist to create topological structure in the workspace, there is only one region/sub-graph (the entire spatial lattice) in this scenario, as shown by Figure 7.8.

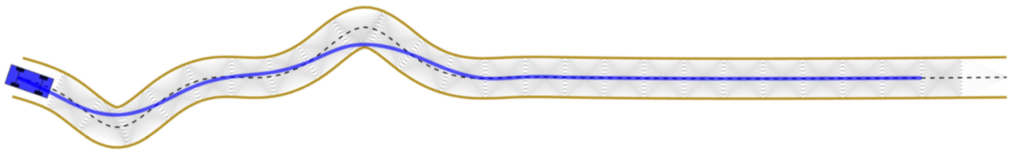


Figure 7.9: Reference smoothing & nudging snapshot for scenario I.

Reference Smoothing & Nudging Figure 7.9 illustrates the generated blue reference plan. Compared to the black dashed centerline, the blue reference is smoother by "cutting" corners like a human driver. This helps reduce the overall steering effort, and allows a higher speed in traversing this curvy road segment.

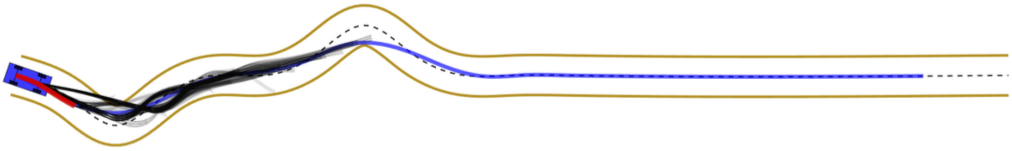


Figure 7.10: Spatiotemporal maneuver pattern analysis snapshot for scenario I.

Spatiotemporal Maneuver Pattern Analysis A pool of fixed time-horizon trajectories are sampled around the generated reference. These trajectories are spatially more near-term compared to the reference. Without any obstacles, all valid trajectories belong to the one and only maneuver pattern, as shown in Figure 7.10. The red trajectory is the best-scored one from this group of sampled trajectories, and will be used as the seeding trajectory, based on the cost function defined for the subsequent trajectory optimization module.

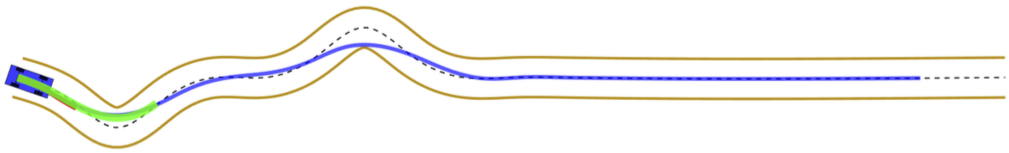


Figure 7.11: Trajectory optimization snapshot for scenario I.

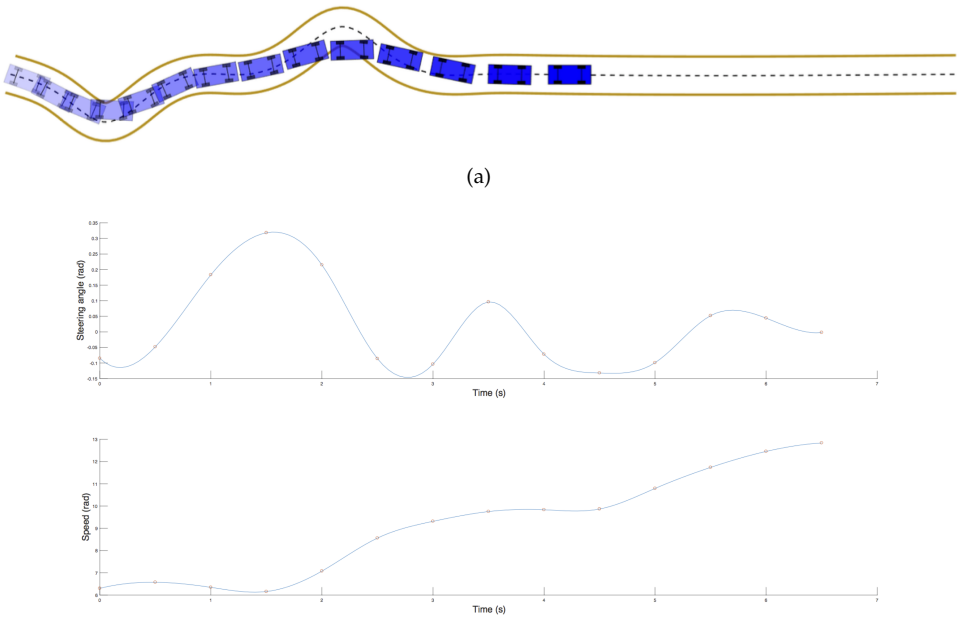
Trajectory Optimization Starting from the red seeding trajectory generated above, the green locally optimized trajectory simply traces the blue reference plan, as shown in Figure 7.11.

Full Scenario Handling The full scenario maneuver sequence is demonstrated in Figure 7.12(a) with the steering and speed plots shown in Figure 7.12(b). It can be seen that the full handling takes care of the high-curvature regions by "cutting" corners like a human and we can traverse the full lane segment efficiently.

7.3.2 Scenario II: Curve with Obstacle Pair

Figure 7.13 demonstrates the second scenario, in which two stationary obstacles are placed on a slightly curved road segment. The goal is to navigate through this segment while smoothly swerving around these obstacles.

Spatial Region Segmentation Due to the existence of the obstacles, four segmented graphs reflecting the topologically different patterns are illustrated in Figure 7.14.



(a)

(b)

Figure 7.12: The full scenario handling overlay plot in 7.12(a) and the steer-speed history plot in 7.12(b).

Figure 7.13: Scenario II

They correspond to swerving right-left (a), right-right (b), left-left (c) and left-right (d), where the first direction represents the side on which the trajectory swerves around the first obstacle in Figure 7.13, and the second direction represents the same for the second obstacle. By inspecting the patterns in Figure 7.14, we can see that the easiest-to-traverse segmentation is (d), in which most of the full lattice is preserved.

Reference Smoothing & Nudging Planning on the sub-graph of region (d), the result of reference nudging can be obtained and demonstrated in Figure 7.15. The reference path is nudged slightly to the side of the stationary obstacles while otherwise tracking the centerline.

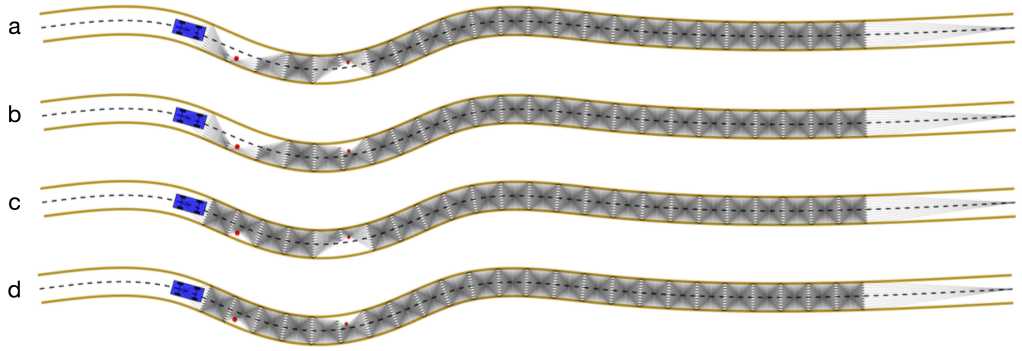


Figure 7.14: Spatial region segmentation snapshot for scenario II.

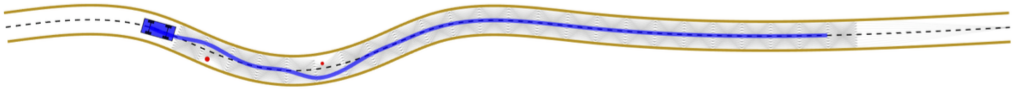


Figure 7.15: Reference smoothing & nudging snapshot for scenario II.

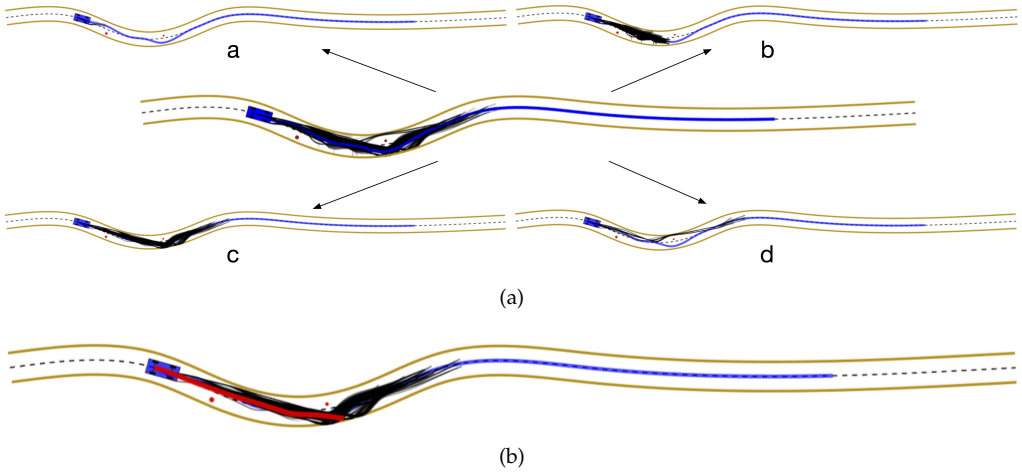


Figure 7.16: Spatiotemporal maneuver pattern analysis snapshot for Scenario II.

Spatiotemporal Maneuver Pattern Analysis As described in Section 7.2.4, linear spatiotemporal edges are generated by placing time-layers of nodes along the reference with small lateral offset, and connecting them across time-layers. But since the edges may have a large spatial gap, even if the nodes are sampled laterally close to the reference, due to the road curve, it is possible to re-generate a trajectory that violates the

spatial topological pattern already determined above (group d in 7.16(a), which in this case violates the pattern by swerving to the left of the second obstacle).

After removing topology-inconsistent group d, we can see that the group c makes the most progress longitudinally within the fixed temporal horizon of the trajectory. Therefore, we choose the best-scored red trajectory within this group as the seeding trajectory for subsequent trajectory optimization, as shown in 7.16(b).

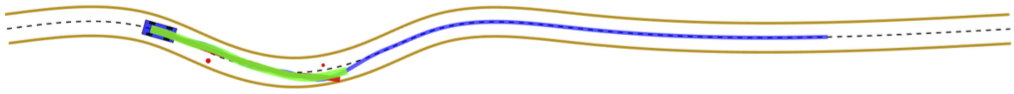


Figure 7.17: Trajectory optimization snapshot for scenario II.

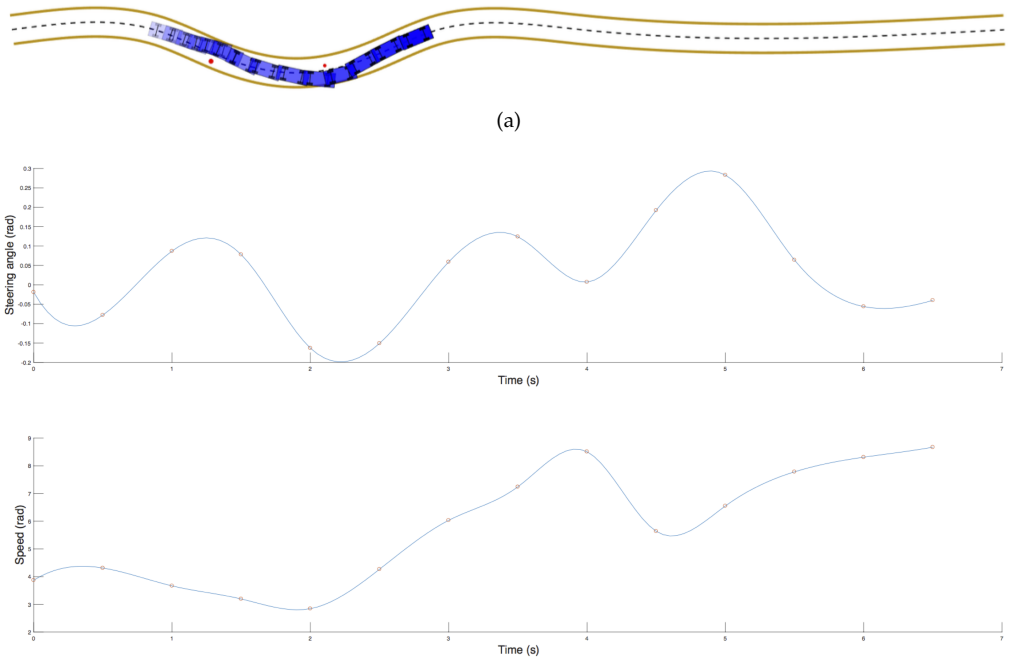
Trajectory Optimization Since the blue reference is only kinematically feasible, the optimization process starts from the red seeding trajectory, and converges to the green optimized trajectory, which deviates slightly from the reference to comply with constraints, as shown in Figure 7.17.

Full Scenario Handling Figure 7.18 shows the full handling sequence. The overall command sequence looks smooth and feasible while still preserving the gist of the reference plan shown in Figure 7.15.

7.3.3 Scenario III: Bicyclist + Pedestrian

In this scenario, we demonstrate the in-lane cruising capability with multiple moving obstacles' interference. In Figure 7.19, the red circular object represents a pedestrian moving vertically upward, while the thin red rectangle represents a bicyclist moving from left to right, i.e., along the road. The goal is to show a bicyclist overtaking behavior while avoiding the crossing pedestrian.

Spatial Region Segmentation Since no stationary obstacles exist and the spatial region segmentation module does not consider moving obstacles, there is only one region/sub-graph (the entire spatial lattice) in this scenario, as shown by Figure 7.20.



(a)

(b)

Figure 7.18: Full scenario handling overlay for scenario II.

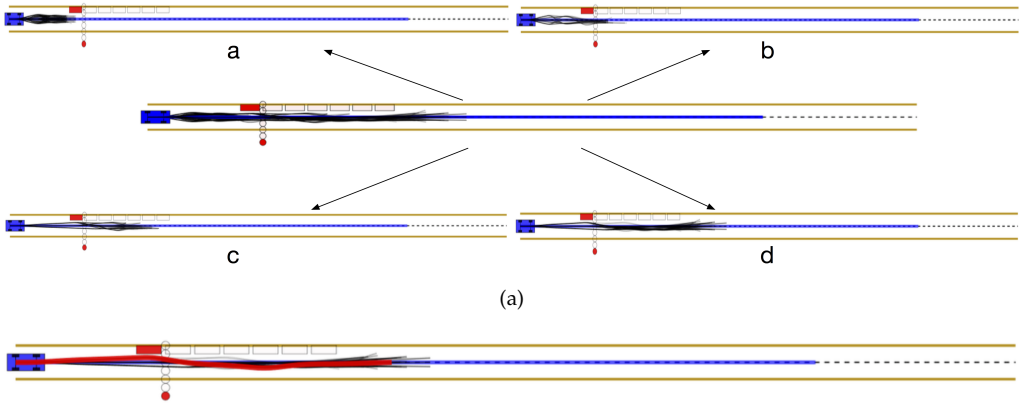
Figure 7.19: Scenario III

Figure 7.20: Segmented spatial region plot for scenario III.

Reference Smoothing & Nudging Figure 7.21 illustrates the generated blue reference plan. Since the centerline is straight, the reference is simply a straight segment up to the spatial look-ahead horizon.

Spatiotemporal Maneuver Pattern Analysis Since all the sampled trajectories have the same temporal look-ahead horizon, their speeds determine their spatiotemporal trace, which will further lead to both region-based and topology-based distinctions,

Figure 7.21: Reference smoothing & nudging snapshot for scenario III.



(a)

(b)

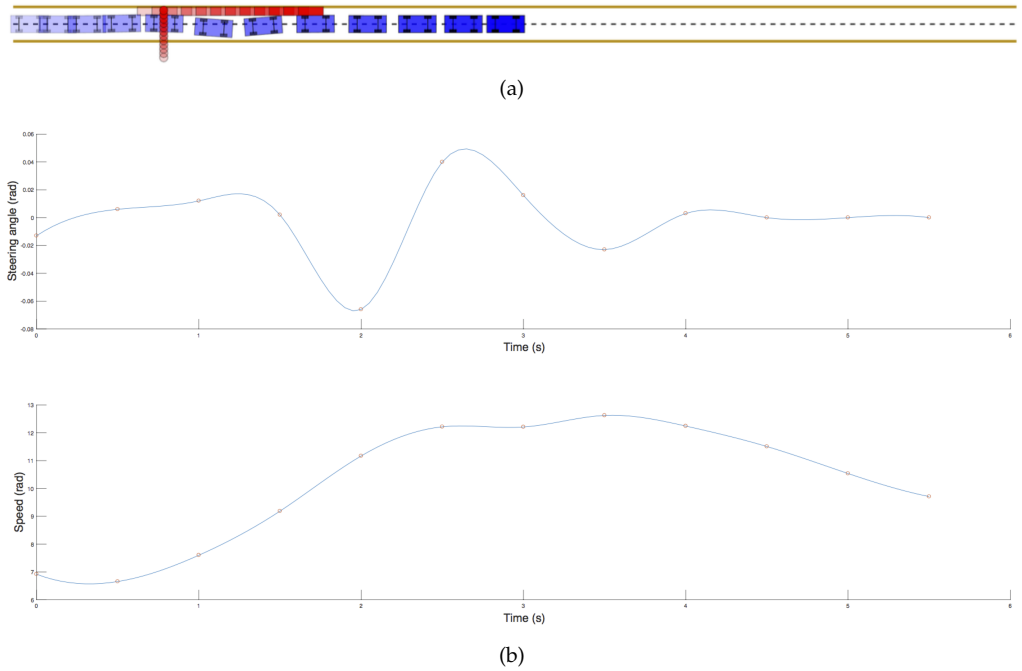
Figure 7.22: Spatiotemporal maneuver pattern analysis snapshot for Scenario III.

as described in Section 5.3. In Figure 7.22(a), group a represents the subset of the sampled trajectories that passes neither pedestrian nor bicyclist given the temporal horizon; group b represents the subset that passes to the right of the pedestrian right while following the leading bicyclist; group c represents the subset that passes to the left of the pedestrian while following the leading bicyclist; group d represents the subset that passes between the objects (i.e., to the left of the pedestrian and to the right of the bicyclist).

From these identified four groups, we prefer the one that makes the most progress spatially and achieves passing maneuvers for both. Therefore, group d is chosen and the best-scored trajectory within it is selected as the seeding trajectory for the subsequent optimization, as shown in 7.22(b).

Figure 7.23: Trajectory optimization snapshot for scenario III.

Trajectory Optimization The optimization process starts from the red seeding trajectory, and converges to the green optimized trajectory, which performs the same high-level behavior but with improved (reduced) control efforts, as shown in Figure 7.23.



(a)

(b)

Figure 7.24: Full scenario handling overlay for scenario III.

Full Scenario Handling Figure 7.24 shows the full handling sequence. The APV accelerates to overtake the pedestrian before it gets into the center of the lane, and further overtakes the bicyclist in front.

7.3.4 Scenario IV: Vehicle Follow with Obstacle Pair

Figure 7.25: Scenario IV

In this scenario, we demonstrate vehicle following with occasional static obstacles. In Figure 7.25, the red rectangle represents a leading moving vehicle with two stationary obstacles represented by red circular disks. The goal is to safely follow the leading vehicle while performing stationary obstacle avoidance.

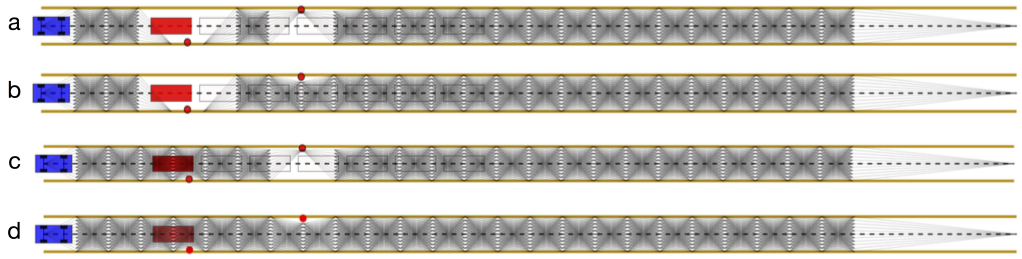


Figure 7.26: Spatial region segmentation snapshot for scenario IV.

Spatial Region Segmentation Disregarding the moving obstacle, this scenario is very similar to that of Scenario II explained in Section 7.3.2. Four sub-graphs are segmented to reflect the topologically different patterns in Figure 7.26. They correspond to swerving right-left (a), right-right (b), left-left (c) and left-right (d), where the first direction represents the side on which the trajectory swerves around the first obstacle, and the second direction represents the same for the second obstacle. Obviously, the easiest-to-traverse segmentation is sub-graph d, in which most of the full lattice is preserved, corresponding to a reference that swerves in between the two stationary obstacles.

Figure 7.27: Reference smoothing & nudging snapshot for scenario IV.

Reference Smoothing & Nudging Planning on the sub-graph d yields the blue reference plan shown in Figure 7.27. The reference path is nudged slightly to the side of the stationary obstacles while tracking the centerline for the most part.

Spatiotemporal Maneuver Pattern Analysis The trajectory pool is sampled spatially close to the reference as shown in Figure 7.28(a). The valid trajectories in black avoid

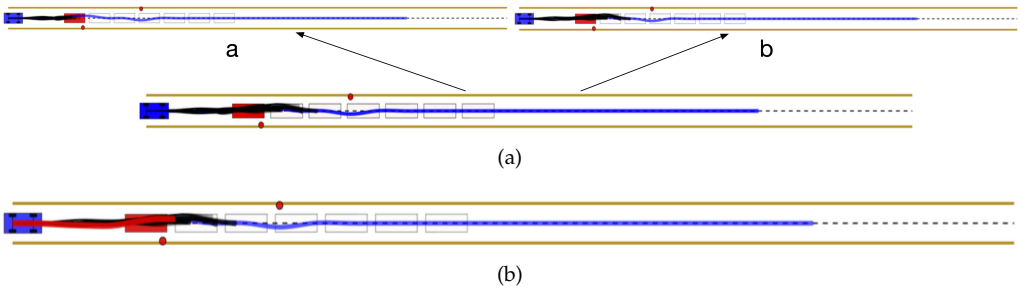


Figure 7.28: Spatiotemporal maneuver pattern analysis snapshot for Scenario IV.

interference with the red leaving vehicle which has moved down the road far enough. The large moving vehicle in the lead removes all vehicle overtake possibilities. Therefore, only two trajectory groups were identified based on where the fixed-horizon trajectory terminates. We prefer the spatially most progressive group b, and choose the best-scored red trajectory within this group as the seeding trajectory for the optimization next as shown in Figure 7.28(b).

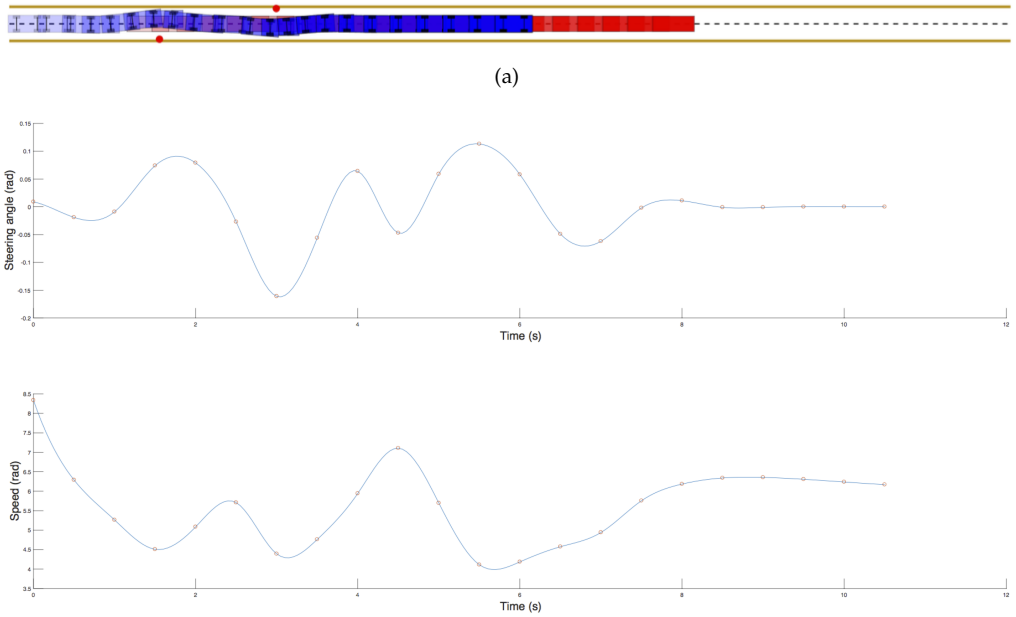
Figure 7.29: Trajectory optimization snapshot for scenario IV.

Trajectory Optimization The optimization process starts from the red seeding trajectory, and converges to the green optimized trajectory as shown in Figure 7.29.

Full Scenario Handling Figure 7.30 shows the full handling sequence. The APV smoothly avoids the stationary obstacles at lower speed, and resumes the following behavior after the avoidance maneuver.

7.4 Comparison to State of the Art

Due to the lack of available implementations of other planners, it is not easy to directly compare head-to-head against other works. Therefore, we will highlight the features that are vital to robust on-road driving and compare our work against the



(a)

(b)

Figure 7.30: Full scenario handling overlay for scenario IV.

prior planners based on their claimed benefits in Table 7.1 before providing an in-depth explanation of the features. The important features are listed below:

- **Nonholonomic**: Applicability to nonholonomic car-like robot.
- **On-road**: Applicability to on-road path/trajectory planning.
- **Deliberative**: Deliberative planning quality.
- **Spatiotemporal**: Explicit spatiotemporal trajectory planning.
- **Tactical**: Motion-level tactical reasoning capability with topological awareness.
- **Hybrid**: Global awareness while converging to a local minimum.

The first two features **Nonholonomic** and **On-road** are self-explanatory. We therefore focus our discussion over the other four features.

No #	Methods	Nonholonomic	On-road	Deliberative	Spatiotemporal	Tactical	Hybrid
M1	Boss's Local Planner [Ferguson et al., 2008]	✓	✓	×	✓*	×	×
M2	Junior's RNDF Follower [Montemerlo et al., 2008]	✓	✓	✓	✓*	×	×
M3	Nonholonomic Potential field [Huang et al., 2006]	✓	×	×	×	×	×
M4	Biased RRT [Kuwata et al., 2008]	✓	✓	✓	✓	×	×
M5	Spatiotemporal lattice [Ziegler and Stiller, 2009, McNaughton, 2011]	✓	✓	✓	✓	×	×
M6	Timed Elastic Band [Roesmann et al., 2012]	✓	×	✓	✓	×	×
M7	Hybrid-A* + Conjugate Gradient Descent Smoothing [Dolgov et al., 2008]	✓	×	✓	×	×	✓
M8	iLQR/DDP [Li and Todorov, 2004b, van den Berg, 2016, Tassa et al., 2014]	✓	×	✓	✓	×	×
M9	Cell decomposition + Mixed-Integer Programming [Park et al., 2016]	✓	✓*	✓	×	✓*	✓
M10	Homotopy marker function + Timed Elastic Band [Rösmann et al., 2015]	✓	×	✓	✓	✓*	×
P	**Proposed**	✓	✓	✓	✓	✓	✓

Table 7.1: Comparison to the state-of-the-art in the highlighted features.

Deliberative: The deliberative planning quality is important for an APV. To better understand what "deliberative" means in this context, it is useful to remember the classical reactive control approaches reviewed in Section 2.2.1, which directly generate a control signal without reasoning about the predicted evolution of the world.

In contrast, a deliberative approach explicitly takes into account a realistic prediction of the world evolution, and yields a trajectory plan that is not an instantaneous control signal, but a sequence of states/control with a certain spatial and temporal horizon. Oftentimes, a deliberative approach can be viewed as "reactive" if such an horizon is too short. In other words, the duration of the temporal planning horizon is a practical indicator of the level of deliberativeness. On the other hand, planning

over too long an horizon should also be avoided, since it is not only computationally inefficient, but also unnecessary.

In table 7.1, M1 sampled too short-sighted trajectories, and M3 used a control-based avoidance approach. The other planners listed all possess a certain amount of deliberativeness. The next feature makes a distinction between spatial deliberativeness and a more comprehensive spatiotemporal deliberativeness.

Spatiotemporal: Spatiotemporal (trajectory) planning capability stands in contrast to the spatial-alone (path) planning or temporal-alone (speed) planning, whose search spaces are significantly more restricted. This property is important for on-road driving, since the APV typically operates in a dynamic environment, where both the path and speed should be adjusted simultaneously to respond properly to such environment disturbances.

In Table 7.1, M1 and M2 are semi-checked because they do not have rich speed variations in their local trajectory samples. M3 is not checked since it is a control-based method. M7 does not have this property since it used pure path planning/smoothing. M9 performs cell-decomposition and optimization in a given snapshot, rather than the full prediction over a time horizon, which is equivalent to pure path planning.

Tactical: Motion-level tactical reasoning with topological awareness This property allows the trajectory planner to propose certain discrete decisions related to the motion itself, e.g., whether to pass an object or not and from which side. The benefit is that the planner becomes more knowledgeable about the environment, and these proposals are tightly coupled to the planning dynamics to provide better feasibility guarantees. From a pure planning solver's perspective, this property allows reduction of the search space for sampling/search-based methods or provision of correct seeding for optimization-based methods.

In Table 7.1, most planners M1 to M8 do not have this property, since the goal of their solvers is purely to reduce cost. M9 and M10 are semi-checked, because they

are only capable of drawing distinctions caused by environment topology in a spatial domain, rather than drawing further region-based and overtaking-sequence-based distinctions in the spatiotemporal domain.

Hybrid: Global awareness while converging to a local minima This property represents the gist of the hybrid sampling-/optimization-based planner. It augments sampling-based planners with the ability to converge to local minima, while it augments the optimization-based planner with global awareness to avoid getting stuck at the wrong local minimum.

In Table 7.1, most planners do not possess this property, with the exception of M7 and M9, both of which used a hybrid planning system.

The proposed approach In summary, our proposed method is the most complete in addressing each of these important features. We obviously has features **Nonholonomic** and **On-road**. It also has feature **Deliberative**. The deliberative planning quality is provided by having a long horizon (greater than 100 meters spatially or 5 seconds temporally) in all four planning modules in Figure 7.3.

Regarding **Spatiotemporal**, the maneuver pattern analysis and focused trajectory optimization modules in our proposed planning system both operate directly in the spatiotemporal domain: the former yields a spatiotemproal segmentation (over the sampled trajectory, as a maneuver pattern), while the latter directly modifies the spatiotemporal trajectory to yield the final trajectory plan.

For feature **Tactical** and **Hybrid**, the proposed method uses search-based and sampling-based methods to explore the high-level decision making alternatives, and make a decision based on a set of high-level features before locally refining with trajectory optimization techniques. This two-step planning methodology endows the planner with the ability to not only be globally aware of the search space of interest, but also explicitly reason about the motion-level maneuver pattern.

Chapter 8

Conclusion

A capable trajectory planner plays a crucial role undergirding the capabilities of an autonomous passenger vehicle (APV). Current trajectory planning methods that rely only on a sampling-based or optimization-based approach have various shortcomings that we reviewed in depth in Chapters 1 and 2. In this thesis, we propose a novel trajectory planning architecture as well as several novel/adapted planning algorithms to achieve a deliberative, tactical, globally-aware and locally optimal trajectory planning system. The contributions and future work are summarized below.

8.1 Contributions

8.1.1 A Hybrid Trajectory Planning Framework

Sampling-based and optimization-based planning methods have both strengths and limitations. We presented a hybrid type of trajectory planning framework composed of four modules that combine the sampling-based and optimization-based trajectory planner approaches in order to maintain the key advantages of both while reducing their limitations. More specifically, we retain the global awareness of the sampling-based planner as well as the ability to converge to a local optimum of the optimization-based method.

8.1.2 Search over Edge-augmented Graph for Reference Path Planning

In a typical directed acyclic graph (DAG), it is easy to calculate cost terms for either the node or the edge; however, it is non-trivial to represent a cost term that requires a pair of neighboring edges (or a tuple of three consecutive nodes). We therefore presented a novel type of augmented graph by constructing augmented nodes with incoming and outgoing edges for an original node over a standard directed acyclic graph (DAG). With the help of this augmented graph, we can easily calculate these cost terms and assign them to the corresponding augmented-node/edge. The augmented graph of a DAG is still a DAG. Therefore, an efficient dynamic programming-based search algorithm can be used to efficiently solve such a graph search problem.

8.1.3 Topological Backward Propagation Algorithm for Region (Sub-Graph) Segmentation

In reference planning, we create a lattice-like DAG that represents the discretized spatial configuration space over which to search. It is useful to segment this configuration space into several sub-regions (or sub-graphs) that encode topologically distinct maneuver patterns. We therefore presented a topological information backward-propagation algorithm for the purpose of region segmentation over a DAG. It takes advantage of some useful homology invariant properties to fit the propagation procedure into a dynamic-programming-like recursive process, and can therefore be efficiently implemented. This technique is applicable to any DAG, e.g., a DAG that spans the spatiotemporal domain. But for spatiotemporal planning, we rely on the following contribution.

8.1.4 Trajectory Sampling-based Maneuver Pattern Analysis for On-Road Driving

In spatiotemporal on-road trajectory planning, different maneuver patterns exist such that pure topological awareness is insufficient to distinguish them all. Therefore, we

proposed a trajectory sampling-based maneuver pattern analysis procedure that can distinguish region-based, topology-based and overtaking-sequence-based differences, thereby providing extra semantics to distinguish between different trajectory subsets within the full trajectory sample pool.

8.1.5 Adapted Linear Quadratic Regulator (LQR) and Iterative-LQR for Trajectory Smoothing/Optimization

In the proposed planning framework, we often generate a coarse path/trajectory first by performing sampling-based planning before generating the ultimate executable ones. In these situations, we adapted an elegant LQR-based trajectory tracker for smoothing to yield model-feasible trajectories, and further adapted a constrained iterative-LQR trajectory optimizer (first-order differential dynamic programming) to generate an execution-feasible trajectory.

8.1.6 Identification of Useful Cost Function Generation Principles

In many prior sampling-based or optimization-based planners, one of the most important engineering design aspects is to develop the cost terms that shape the behavior of the robot. In the cost design section for the iterative-LQR trajectory optimizer, we provided some useful cost function design principles based on convex optimization theories, and suggested the common sources of non-convexity (oftentimes unavoidable in the motion planning field) and their implications.

8.2 Future Work

In this thesis, we provide a comprehensive framework to combine sampling-based and optimization-based methods. We believe this is a very promising future research direction to retain the benefits of both approaches. In particular, we believe the consideration of topological information in traditional trajectory planning will significantly improve planning outcome by giving the planner certain level of environment under-

standing, which will aid more informative decision making as well as helping focus the search space exploration.

For more specific future efforts, many sub-components in the proposed trajectory planning system can be improved, such as: using more complex vehicle models, modeling the APV/objects with better geometric representations, and speeding up collision checking, as well as many trajectory integration calculations through caching and interpolation techniques.

In addition, more involved cost terms reflecting more realistic behavioral and constraint features for on-road driving may be developed following the proposed cost design principles. And it is straightforward practice to apply supervised machine learning to regress the proper weights for the many cost terms designed in multiple planning modules, which will serve as a principled and automated way of performing parameter tuning.

Finally, the proposed framework is still a structured planning system in that the algorithm flow as well as the cost terms are all artificially designed. It would be particularly interesting to develop trajectory planners that do not have such structure by design, but are acquired via more general learning methods (e.g., deep reinforcement learning).

Bibliography

F. Aurenhammer. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, Sept. 1991. 80

H. H. Bauschke, Y. Lucet, and H. M. Phan. On the convexity of piecewise-defined functions. *arXiv*, 16 Aug. 2014. 82

A. Belyaev, P.-A. Fayolle, and A. Pasko. Signed lp-distance fields. *Comput. Aided Des. Appl.*, 45(2):523–528, 2013. 80, 81

D. P. Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999. 91

J. T. Betts. Survey of numerical methods for trajectory optimization. *J. Guid. Control Dyn.*, 21(2):193–207, 1998. 18, 19

D. M. Bevly, J. Ryu, and J. C. Gerdes. Integrating INS sensors with GPS measurements for continuous estimation of vehicle sideslip, roll, and tire cornering stiffness. *IEEE Trans. Intell. Transp. Syst.*, 7(4):483–493, Dec. 2006. 26

S. Bhattacharya. *Topological and geometric techniques in graph search-based robot planning*. PhD thesis, University of Pennsylvania, 2012. 21, 47, 51

S. Bhattacharya, V. Kumar, and M. Likhachev. Search-Based path planning with homotopy class constraints. In *Third Annual Symposium on Combinatorial Search*. aaai.org, 25 Aug. 2010. 21

J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield. Little ben: The ben franklin racing team's entry in the 2007 DARPA urban challenge. *J. Field Robotics*, 25(9):598–614, 1 Sept. 2008. 8

R. Bott and L. W. Tu. *Differential Forms in Algebraic Topology*. Graduate Texts in Mathematics. Springer New York, 2013. 51

T. Brandt, T. Sattel, and J. Wallaschek. Towards vehicle trajectory planning for collision avoidance based on elastic bands. *Int. J. Veh. Auton. Syst.*, 5(1-2):28–46, 2007. 18

H. Breu, J. Gil, D. Kirkpatrick, and M. Werman. Linear time euclidean distance transform algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(5):529–533, May 1995. 80

O. Brock and O. Khatib. Elastic strips: A framework for integrated planning and execution. In *Experimental Robotics VI*, Lecture Notes in Control and Information Sciences, pages 329–338. Springer London, 2000. 18

M. Campbell, E. Garcia, D. Huttenlocher, I. Miller, P. Moran, A. Nathan, B. Schimpf, N. Zych, J. Catlin, F. Chelarescu, and Others. Team cornell: technical review of the DARPA urban challenge vehicle. *DARPA Urban Chall. Tech. Pap*, 2007. 69

C. I. Connolly, J. B. Burns, and R. Weiss. Path planning using laplace's equation. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 2102–2106 vol.3. ieeexplore.ieee.org, May 1990. 10

L. Davis. Handbook of genetic algorithms. *Handbook of genetic algorithms*, 1991. 17

D. J. Demyen and M. Buro. Efficient triangulation-based pathfinding. *Aaai*, 2006. 21

G. V. den Bergen. A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools*, 4(2):7–25, 1999. 80

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1): 269–271, 1 Dec. 1959. 15

C. Dimitrakakis. Online statistical estimation for vehicle control: A tutorial. *Idiap-RR-13-2006*, 2007. 18

D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Practical search techniques in path planning for autonomous driving. *Ann Arbor*, 1001:48105, 2008. 18, 80, 116

S. E. Dreyfus and A. M. Law. *Art and Theory of Dynamic Programming*. Academic Press, Inc., Orlando, FL, USA, 1977. 56

M. Du, J. Chen, P. Zhao, H. Liang, Y. Xin, and T. Mei. An improved RRT-based motion planner for autonomous vehicle in cluttered environments. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4674–4679. ieeexplore.ieee.org, May 2014. 13

P. Felzenszwalb and D. Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell University, 2004. 80

D. Ferguson, T. M. Howard, and M. Likhachev. Motion planning in urban environments. *J. Field Robotics*, 25(11-12):939–960, 1 Nov. 2008. 116

D. Fox and W. Burgard. The dynamic window approach to collision avoidance. *IEEE Trans. Rob.*, 1997. 10

T. Fraichard and V. Delsart. Navigating dynamic environments with trajectory deformation. *CIT. Journal of Computing and Information*, 2009. 19

E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, Apr. 1988. 80

D. J. Griffiths and R. College. *Introduction to electrodynamics*, volume 3. Prentice Hall Upper Saddle River, NJ, 1999. 47, 53

B. Hamner, S. Singh, and S. Scherer. Learning obstacle avoidance parameters from operator behavior. *Journal of Field Robotics*, 2006. 10

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*, 4(2): 100–107, July 1968. 15, 72

A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2002. 47

E. Hernandez, M. Carreras, and P. Ridao. A comparison of homotopic path planning algorithms for robotic applications. *Rob. Auton. Syst.*, 64:44–58, 2015. 22

W. H. Huang, B. R. Fajen, J. R. Fink, and W. H. Warren. Visual navigation and obstacle avoidance using a steering potential function. *Rob. Auton. Syst.*, 54(4):288–299, 28 Apr. 2006. 10, 116

J. Hurdus, A. Bacha, C. Bauman, S. Cacciola, R. Faruque, P. King, C. Terwelp, P. Currier, D. Hong, A. Wicks, and C. Reinholtz. VictorTango architecture for autonomous navigation in the DARPA urban challenge. *J. Aerosp. Comput. Inf. Commun.*, 5(12): 506–529, 2008. 8, 9

J. hwan Jeon, R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma. Optimal motion planning with the half-car dynamical model for autonomous high-speed driving. In *2013 American Control Conference*, pages 188–193. ieeexplore.ieee.org, June 2013. 13

C.-R. Hwang. Simulated annealing: Theory and applications. *Acta Appl. Math.*, 12(1): 108–111, 1 May 1988. 17

Y. K. Hwang and N. Ahuja. Gross motion planning—a survey. *ACM Computing Surveys (CSUR)*, 1992. 9

D. Jacobson and D. Mayne. *Differential Dynamic Programming*. Elsevier Sci. Publ., 1970. 19, 69

D. H. Jacobson. New second-order and first-order algorithms for determining optimal control: A differential dynamic programming approach. *J. Optim. Theory Appl.*, 2(6): 411–440, 1 Nov. 1968. 19, 69

K. D. Jenkins. The shortest path problem in the plane with obstacles: A graph modeling approach to producing finite search lists of homotopy classes. *IEEE Trans. Rob.*, 1991. 21, 48

S. Kammel, J. Ziegler, B. Pitzer, M. Werling, T. Gindele, D. Jagzent, J. Schröder, M. Thuy, M. Goebl, F. v. Hundelshausen, O. Pink, C. Frese, and C. Stiller. Team AnnieWAY's autonomous system for the 2007 DARPA urban challenge. *J. Field Robotics*, 25(9): 615–639, 1 Sept. 2008. 9

Y. Kanayama and B. I. Hartman. Smooth local path planning for autonomous vehicles. In I. J. Cox and G. T. Wilfong, editors, *Autonomous Robot Vehicles*, pages 62–67. Springer New York, 1990. 11

S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, 30(7):846–894, 1 June 2011. 13

L. E. Kavraki. Computation of configuration-space obstacles using the fast fourier transform. *IEEE Trans. Rob. Autom.*, 11(3):408–413, June 1995. 80

L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Rob. Autom.*, 12(4):566–580, Aug. 1996. 12, 21

A. Kelly and B. Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *Int. J. Rob. Res.*, 22(7-8):583–601, 1 July 2003. 11, 13

O. Khatib. Real-Time obstacle avoidance for manipulators and mobile robots. *Int. J. Rob. Res.*, 5(1):90–98, 1 Mar. 1986. 9

S. Kim, K. Sreenath, S. Bhattacharya, and V. Kumar. Trajectory planning for systems with homotopy class constraints. In J. Lenarcic and M. Husty, editors, *Latest Advances in Robot Kinematics*, pages 83–90. Springer Netherlands, 2012. 22, 50

S. Koenig and M. Likhachev. D* lite. *AAAI/IAAI*, 2002. 15

S. Koenig, M. Likhachev, and D. Furcy. Lifelong planning a*. *Artif. Intell.*, 155(1): 93–146, 1 May 2004. 15

Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1398–1404 vol.2. ieeexplore.ieee.org, Apr. 1991. 10

R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artif. Intell.*, 1985. 15

B. Krogh and C. Thorpe. Integrated path planning and dynamic steering control for autonomous vehicles. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 1664–1669. ieeexplore.ieee.org, Apr. 1986. 9

M. Kuderer, C. Sprunk, H. Kretzschmar, and W. Burgard. Online generation of homotopically distinct navigation paths. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6462–6467. ieeexplore.ieee.org, May 2014. 21

H. Kurniawati and T. Fraichard. From path to trajectory deformation. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 159–164. ieeexplore.ieee.org, Oct. 2007. 19

Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. How. Motion planning in complex environments using closed-loop prediction. In *AIAA Guidance, Navigation and Control Conference and Exhibit*. arc.aiaa.org, 2008. 69, 116

Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How. Real-Time motion planning with applications to autonomous urban driving. *IEEE Trans. Control Syst. Technol.*, 17(5):1105–1118, 2009. 13

S. M. Lavalle. Rapidly-Exploring random trees: A new tool for path planning. *Citeseer*, 1998. 13

S. M. LaValle, M. S. Branicky, and S. R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *Int. J. Rob. Res.*, 23(7-8):673–692, 1 Aug. 2004. 13

D. Le and E. Plaku. Guiding sampling-based tree search for motion planning with dynamics via probabilistic roadmap abstractions. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 212–217. ieeexplore.ieee.org, 2014. 12, 13

J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '90, pages 327–335, New York, NY, USA, 1990. ACM. 10

J. Leonard, D. Barrett, J. How, S. Teller, M. Antone, and others. Team MIT urban challenge technical report. *dspace.mit.edu*, 2007. 8, 13

W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229, 2004a. 69, 75

W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229. newmaeweb.ucsd.edu, 2004b. 19, 91, 116

M. Likhachev and D. Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *Int. J. Rob. Res.*, 28(8):933–945, 1 Aug. 2009. 14

M. Likhachev, G. J. Gordon, and others. ARA*: Anytime a* with provable bounds on sub-optimality. *Adv. Neural Inf. Process. Syst.*, 2003. 15

F. Lingelbach. Path planning using probabilistic cell decomposition. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 1, pages 467–472 Vol.1. ieeexplore.ieee.org, Apr. 2004. 21

T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, Oct. 1979. 21

D. G. Luenberger. *Introduction to linear and nonlinear programming*, volume 28. Addison-Wesley Reading, MA, 1973. 40

M. McNaughton. *Parallel Algorithms for Real-time Motion Planning*. PhD thesis, Carnegie Mellon University, 2011. 14, 37, 77, 97, 102, 116

J. B. M. Melissen and P. C. Schuur. Covering a rectangle with six and seven circles. *Discrete Appl. Math.*, 99(1–3):149–156, 1 Feb. 2000. 80

I. Miller, M. Campbell, D. Huttenlocher, F.-R. Kline, A. Nathan, S. Lupashin, J. Catlin, B. Schimpf, P. Moran, N. Zych, E. Garcia, M. Kurdziel, and H. Fujishima. Team cornell's skynet: Robust perception and planning in an urban environment. *J. Field Robotics*, 25(8):493–527, 1 Aug. 2008. 8

M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun. Junior: The stanford entry in the urban challenge. *J. Field Robotics*, 25(9):569–597, 1 Sept. 2008. 1, 8, 116

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

V. Narayanan, P. Vernaza, M. Likhachev, and S. M. LaValle. Planning under topological constraints using beam-graphs. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 431–437. ieeexplore.ieee.org, May 2013. 21

National Highway Traffic Safety Administration. Traffic safety facts 2011 data–pedestrians. *Ann. Emerg. Med.*, 62(6):612, Dec. 2013. 1

Park, Junghee, and Ph. D. Massachusetts Institute of Technology. *A homotopy-based hierarchical framework for semi-autonomous/autonomous vehicle navigation*. PhD thesis, Massachusetts Institute of Technology, 2016. 21, 116

J. Park, S. Karumanchi, and K. Iagnemma. Homotopy-Based Divide-and-Conquer strategy for optimal trajectory planning via Mixed-Integer programming. *IEEE Trans. Rob.*, 31(5):1101–1115, Oct. 2015. 21

A. Piazzi and C. Bianco. Quintic G 2-splines for trajectory planning of autonomous vehicles. *, 2000. IV 2000. Proceedings of the IEEE*, 2000. 11

P. Pirjanian. Behavior coordination mechanisms-state-of-the-art. *Citeseer*, 1999. 9

M. Pivtoraiko, R. A. Knepper, and A. Kelly. Differentially constrained mobile robot motion planning in state lattices. *J. Field Robotics*, 26(3):308–333, 1 Mar. 2009. 13, 37

S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, pages 802–807 vol.2. ieeexplore.ieee.org, May 1993. 18

N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich. Maximum margin planning. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 729–736, New York, NY, USA, 2006. ACM. 103

C. Roesmann, W. Feiten, T. Woesch, F. Hoffmann, and T. Bertram. Trajectory modification considering dynamic constraints of autonomous robots. In *ROBOTIK 2012; 7th German Conference on Robotics*, pages 1–6. ieeexplore.ieee.org, May 2012. 19, 116

C. Rösmann, F. Hoffmann, and T. Bertram. Planning of multiple robot trajectories in distinctive topologies. In *Mobile Robots (ECMR), 2015 European Conference on*, pages 1–6, 2015. 22, 116

W. Rudin. Real and complex analysis. *Real and complex analysis*, 1987. 21

E. Schmitzberger, J. L. Bouchet, M. Dufaut, D. Wolf, and R. Husson. Capture of homotopy classes with probabilistic road map. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2317–2322 vol.3. ieeexplore.ieee.org, 2002. 21

D. Silver, J. Andrew Bagnell, and A. Stentz. Applied imitation learning for autonomous navigation in complex natural terrain. In A. Howard, K. Iagnemma, and A. Kelly, editors, *Field and Service Robotics*, Springer Tracts in Advanced Robotics, pages 249–259. Springer Berlin Heidelberg, 2010. 103

J. M. Snider. Automatic steering methods for autonomous automobile path tracking. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009. 31, 69, 70

M. Sniedovich. Dijkstra's algorithm revisited: the dynamic programming connexion. *Control Cybernet.*, 2006. 15

J. A. Stratton, I. Antennas, and P. Society. *Electromagnetic Theory*. An IEEE Press classic reissue. Wiley, 2007. 21

Y. Tassa, N. Mansard, and E. Todorov. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175. ieeexplore.ieee.org, May 2014. 20, 69, 116

M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, 46(3):362–394, May 1999. 15

M. Thorup. Integer priority queues with decrease key in constant time and the single source shortest paths problem. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 149–158, New York, NY, USA, 2003. ACM. 15

S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niek-

erk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Stanley: The robot that won the DARPA grand challenge. *J. Field Robotics*, 23(9):661–692, 1 Sept. 2006. 11, 18

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.*, pages 300–306 vol. 1. ieeexplore.ieee.org, June 2005. 69

C. Urmson and R. G. Simmons. Approaches for heuristically biasing RRT growth. *IROS*, 2003. 13

C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y.-W. Seo, S. Singh, J. Snider, A. Stentz, W. r. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson. Autonomous driving in urban environments: Boss and the urban challenge. *J. Field Robotics*, 25(8):425–466, 1 Aug. 2008. 1, 6, 8, 11, 14, 77, 96

J. van den Berg. Extended LQR: Locally-Optimal feedback control for systems with Non-Linear dynamics and Non-Quadratic cost. In M. Inaba and P. Corke, editors, *Robotics Research*, Springer Tracts in Advanced Robotics, pages 39–56. Springer International Publishing, 2016. 19, 69, 116

O. Von Stryk and R. Bulirsch. Direct and indirect methods for trajectory optimization. *Ann. Oper. Res.*, 37(1):357–373, 1992. 18

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)*, pages 763–770, June 2013. 69, 96

F. A. Wilson and J. P. Stimpson. Trends in fatalities from distracted driving in the united states, 1999 to 2008. *American journal of public health*, 100(11):2213–2219, 2010. 1

B. M. Yu, K. V. Shenoy, and M. Sahani. Derivation of kalman filtering and smoothing equations. *Saatavissa: http://wwwnpl. stanford. edu/\byronyu/papers/derive_ks. pdf. Hakupäivä*, 7:2010, 2004. 19

J. Ziegler and C. Stiller. Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1879–1884. ieeexplore.ieee.org, Oct. 2009. 14, 116

J. Ziegler and C. Stiller. Fast collision checking for intelligent vehicle motion planning. In *2010 IEEE Intelligent Vehicles Symposium*, pages 518–522. ieeexplore.ieee.org, June 2010. 80

J. Ziegler, P. Bender, T. Dang, and others. Trajectory planning for bertha—a local, continuous method. *2014 IEEE Intelligent*, 2014. 18, 77