Lattice-Based Motion Planning with Optimal Motion Primitives

by

Alexander Botros

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2021

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Emilio Frazzoli Professor, Dept. of Mechanical and Process Engineering, Swiss Federal Institute of Technology in Zürich (ETH Zürich)
Supervisor:	Stephen L. Smith Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal Member(s):	Andrew Heunis Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
	Mahesh Tripunitara Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal-External Member:	William Melek Professor, Dept. of Mechanical and Mechatronics Engineering, University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

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

Abstract

In the field of navigation for autonomous vehicles, it is the responsibility of a local planner to compute reference trajectories that are then be followed by a tracking controller. These trajectories should be safe, kinematically feasible, and optimize certain desirable features like low travel time and smoothness/comfort. Determining such trajectories is known as the motion planning problem and is the focus of this work.

In general, the motion planning problem is intractable, and simplifications must be made in order to compute reference trajectories quickly and in real time. A common strategy involves adopting a simple kinematic model for the trajectory. However, overly simplified models can lead to references that are infeasible for the vehicle. These are hard for a tracking controller to follow resulting in large tracking error and frequent re-planning. In contrast, lattice-based motion planning simplifies the motion planning problem by restricting the set of allowable motions. In detail, lattice-based motion planning works by discretizing the configuration space of a vehicle into a regularly repeating grid called a lattice. The set of all optimal feasible trajectories between vertices of this lattice are *pre*-computed and a subset called a control set is selected. Trajectories of this pre-computed subset are then joined together online to form more complex compound maneuvers. Because trajectories between lattice vertices are pre-computed, the complexities of the motion planning problem are considered offline. While not every trajectory is available to a lattice-based planner, every trajectory that *is* available is feasible and optimal.

Selecting a control set is an important step in lattice-based motion planning since the optimality of each element of the control set does not guarantee the optimality of compound maneuvers. These control sets are often selected based on intuition and experience. Broadly, the size of a control set has a positive effect on the quality of computed trajectories, but at the expense of run time performance. A control set is said to $t-$span a lattice if trajectories between lattice vertices can be approximated to within a factor of $t \geq 1$ as compound maneuvers of elements of the control set. Given an acceptable allowance $t \geq 1$ on the sub-optimality of compound maneuvers in a lattice, the problem of computing the smallest control set that $t-$spans the lattice is called the *minimum t-spanning control set problem*. In essence, this problem seeks to optimize a trade-off between the quality of compound maneuvers and the time required to compute them.

This work details solutions and applications of the minimum $t-$spanning control set problem in autonomous vehicle navigation. In particular, we first investigate an instance of the problem that can be solved efficiently, provide an intuitive solution, and outline the applications of this instance in the field of any-angle path planning in a two dimensional environment.

Next, we provide a novel method to compute trajectories that optimize an adjustable trade-off between certain desirable features. The relative importance of each of these features may differ by user, and the techniques developed here are able to reflect these preferences. The NP-completeness of the general minimum $t-$spanning control set problem is established here, and we present a mixed integer linear program that encodes the problem. The trajectories we propose in conjunction with the mixed integer linear program, result in a method to compute a minimum $t-$spanning control set whose elements are kinematically feasible and reflect the preferences of a user if those preferences are known.

Finally, we consider the problem of simultaneously learning the preferences of a single user from demonstrations and computing sparse control sets for that user. We propose a technique to solve this problem that leverages a separation principle: first estimate the preferences of the user based on demonstrations, then compute a control set of trajectories that are optimal given the estimated preferences. We show that this approach optimally solves the problem. Combining the work of this thesis results in a method by which tailored control sets that reflect the preferences of a user can be determined from the demonstrations of that user. These control sets have the following beneficial attributes: 1) each element of the control set is optimal for the estimated preferences of the user, and 2) the control set optimizes a trade-off between the quality of compound maneuvers between lattice vertices – as defined by the estimated preferences of the user – and time required to compute them.

Acknowledgements

First and foremost, thank you to my wonderful family. This includes, but is not limited to, my wife Carolyn, son Peter, mother Sue, brother Will, sister Jenn, as well as all of their myriad children and spouses. And my dog Watson. I would also like to thank Dr. Stephen Smith whose patience and guidance made this work possible.

A very special thank you must be given to my good friend and collaborator Nils Wilde who gave me music, friendship and things to think about while I studied far from home.

Finally, I would like to wholeheartedly thank NSERC for their support of my research.

To all of those I mentioned above and more, I hope I can earn the wonderful support you've given me.

Dedication

This thesis, like all of my research, is dedicated to my Dad Dr. Adel Botros. Life doesn't last long. Math does. Thank you for both.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

CCR Continuous Curvature Rate 11, 12, 61

HCR Hybrid Curvature Rate 11, 12, 61

IS Integral Squared. Refers to the integral of the squared magnitude of a function along a trajectory 80

MILP Mixed Integer Linear Programming or Mixed Integer Linear Program x, xi, 5, 8, 90, 91, 94, 95, 97, 100, 107, 108, 117, 124–126, 134

MSKCS Minimum Spanning K-Control Set xiv, 33, 34, 117, 123–125, 129–131

MTSCS Minimum t-Spanning Control Set viii–x, 5–10, 16, 18, 26, 30, 32–39, 50, 51, 55, 58, 90–94, 98, 102, 125, 133

TPBV Two Point Boundary Value 2, 12, 14, 15

Chapter 1

Introduction

As autonomous vehicles and car-like robots (which we also refer to as vehicles) become increasingly important to modern society [42], so too does the need to develop safe, feasible, possibly comfortable trajectories for them. In the field of motion planning, this need is embodied by motion planning problem. At its highest level, the motion planning problem is to compute trajectories that adhere to the kinematic and physical constraints of the vehicle while simultaneously maximizing desirable properties like low travel times and comfort [78]. During the course of a vehicles' movement, it is the role of a global planner

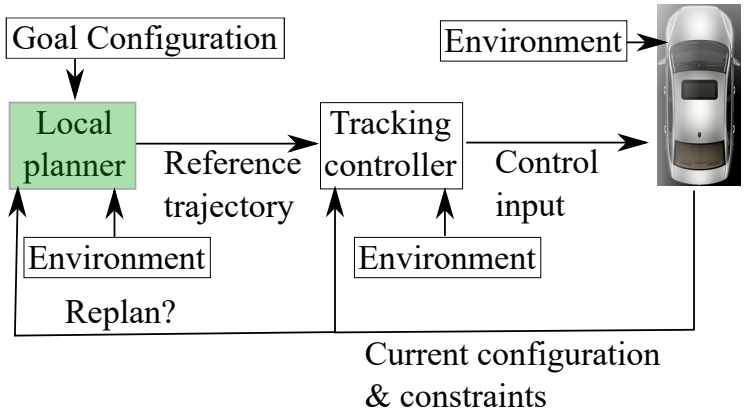


Figure 1.1: The role of the local planner in autonomous driving.

to compute an initial collision-free trajectory between start and goal configurations that does not take into account the differential constraints of a vehicle [116]. This trajectory – expressed as a sequence of global waypoints – is then refined by a local planner by solving

the motion planning problem between waypoints. The result is a reference trajectory that can be followed by a tracking controller [108]. The tracking controller computes a control signal that allows the vehicle to follow reference. The process of computing and following a reference trajectory – summarized in Figure 1.1 – is performed iteratively with the local planner computing a new reference from the vehicles' current configuration to its current waypoint goal in accordance with a re-planning rule [108].

This work focuses on the local planner, in particular on the motion planning problem. In general, this problem is intractable [3] owing in part to potentially complex vehicle models which impede the calculation of Two Point Boundary Value (TPBV) problems. That is, it may be difficult to compute *any* kinematically feasible reference trajectory for the vehicle, let alone one that maximizes desirable properties. Therefore, simplifying assumptions must be made so that the local planner can quickly develop reference trajectories online.

Overly simplified kinematic models result in infeasible reference trajectories [34] and large tracking error [29, 64] that may necessitate frequent re-planning. In contrast, solving TPBV problems online for complex models is computationally impractical. Lattice-based motion planning simplifies the motion planning problem by limiting the allowable trajectories. In detail, lattice-based planners works by discretizing the space of all possible vehicle configurations into a countable set (or *lattice*) of regularly repeating configurations. Kinematically feasible trajectories between lattice configurations (or *vertices*) are *pre*-computed in such a way as to maximize desirable properties. This is done in the absence of obstacles. Finally, a subset of these trajectories called a *control set* is selected [76, 88, 90, 106]. The elements of the control set – called *motion primitives* – can then be concatenated online to form complex compound maneuvers that do not collide with obstacles.

Because vertex to vertex trajectories are pre-computed in lattice-based motion planning, the complexities of the motion planning problem that arise from the kinematic model are considered offline – thus removing the need to solve TPBV problems online. The feasibility of a compound maneuver obtained by concatenating elements of a control set can be guaranteed by ensuring the feasibility of each pre-computed element of the control set. These observations are the foundation of Lattice-based motion planning, one of the most common approaches to solving the motion planning problem [34, 106].

This work focuses two stages of lattice-based motion planning: the development of kinematically feasible trajectories between lattice vertices that maximize desirable properties in the absence of obstacles, and the selection a control set. These two stages: lattice trajectory generation, and control set selection are discussed here:

Lattice Trajectory Generation In differential geometry, a *planar curve* (or just *curve* in this work) is a function from a non-empty interval of $\mathbb{R}$ – called a *parametrization* – to the plane. In this thesis, a *path* is a curve where the chosen parameter is arc-length, while a *trajectory* is a curve parameterized by time. A curve is called C^k continuous if it is continuous up to the k^{th} derivative with respect to its chosen parametrization. On the other hand, the curve is called G^k continuous, or *geometrically* continuous, if there exists a parametrization for which the path is C^k continuous [7]. Typically, a curve is said to be G^k if it is G^k continuous, but not G^{k+1} continuous. Geometric continuity is a measure of smoothness of a curve independent of its parametrization but is equivalent to the familiar notion of C^k continuity when the parameter selected is arc-length [7]. Observe that a trajectory is uniquely defined by a path together with a velocity profile that governs movement along the path through time. That is, the velocity profile offers a re-parametrization of a curve from arc-length to time dictated by the differential equation $ds/dt = v(s)$ for arc-length s, velocity profile $v(s)$, and time t.

In this work, a method to compute trajectories between lattice vertices in the absence of obstacles is presented. This process develops both a path and a velocity profile that optimize a trade-off between smoothness/comfort [78] and travel time [66, 121], though the methods developed here can be used to consider other features as well.

Every path computation technique available in literature will produce a path that is at least G^0 continuous. The paths developed in this work are G^3 continuous. It will be shown in Chapter 4 (Theorem 4.3.1) that this constitutes a necessary condition for the optimality of a G^k-continuous path for many possible cost functions that balance a trad-off between comfort and travel time (e.g. those used in [66, 119, 121, 122]) .

Given a set of weights representing the relative importance of comfort and travel time for a user, the technique presented here iteratively computes and refines a path and a velocity profile to produce a trajectory that best represents the given weights. Critically, because these trajectories can be *pre*-computed and saved for use in lattice-based motion planning, the time required to compute these trajectories is of far less importance than the space required to store them. The methods developed in this work are largely analytical and it will be shown that a trajectory can be easily reconstructed online from only 19 stored constants. This is potentially far less than the number of constants required to describe a trajectory that was derived using purely numerical means. Indeed, a numerically-derived trajectory is described by sampled states. Therefore, to describe a trajectory of length 10 meters with samples taken every 0.1 meters, 300 values would need to be stored: 100 samples $\times$ 3 states per sample (position in $\mathbb{R}^2$ and velocity). In this example, storing a numerically-derived trajectory requires $\approx$ 16 times more values than the proposed method.

In addition to generating trajectories that optimize a *given* trade-off between comfort and travel time (represented as a set of weights), this thesis also describes how to incorporate user demonstrations into the lattice trajectory generation process. In essence, given a set of user demonstrations, a set of weights is computed that best describes the demonstrations. These weights can then be used as inputs to the trajectory generation technique described above, resulting in lattice trajectories that take into account the preferences of a user. Once trajectories between lattice vertices have been computed, a control set is selected. This process is discussed here.

Control Set Selection As a motivating example, consider the motion planning problem of computing the shortest path between start and goal configurations of the form $(x, y) \in \mathbb{R}^2$ when instantaneous changes in heading is permitted without penalization. In such problems – called *any-angle* path planning problems [43, 75, 94] – shortest paths are comprised of a sequence of collision-free straight lines. Consider a discretization of the configuration space $\mathbb{R}^2$ comprised of integer-valued (x, y) pairs arranged in a square lattice. Example control sets for this lattice (red) and paths computed using these control sets (blue) are illustrated in Figure 1.2. It can be shown that in the absence of obstacles, the shortest path using the control set from the left image from any start to any goal in the lattice will be no more than a factor of $\sqrt{2}$ from the length of the shortest path [75]. This error factor is decreased to ≈ 1.08 using the control set in the right image [75]. It

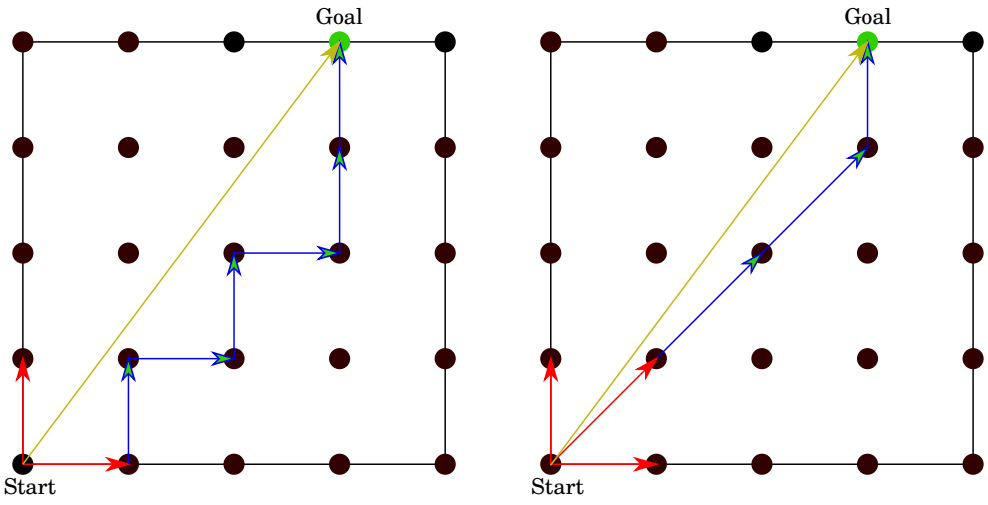


Figure 1.2: Two example control sets (red) and shortest trajectories from start to goal using these control sets (blue). Shortest trajectories are shown in gold.

is observed in [12, 89] that the *number* of primitives in a control set favorably affects the quality of the resulting paths but adversely affects the performance of an online search. This observation motivates the following question: The control sets illustrated in Figure 1.2 result in a maximum error factors of $\sqrt{2}, 1.08$, respectively, but is there any smaller control set that will achieve the same result?

In [89], the authors introduce the notion of a $t-$spanning control set: a control set that guarantees the existence of compound trajectories whose costs are no more than a factor of t from optimal. For example, the control sets in Figure 1.2 will $\sqrt{2}$-span and 1.08-span the lattice, respectively. The authors of [89] propose that an optimal control set for a lattice is one with minimal size that, for a given value $t > 1$, will $t-$span the lattice. Such a set would optimizing a trade-off between path quality and performance. The problem of computing such a control set is called the Minimum t-Spanning Control Set (MTSCS) problem.

The MTSCS problem is, in essence, a coverage problem. In particular, one may think of a trajectory v between lattice vertices as being "covered" by a control set if v can be decomposed into a sequence of trajectories in the control set whose total cost is no more than a factor of t from the cost of v. Thus the MTSCS problem seeks to compute the smallest set that covers a lattice.

As with other coverage problems [55], there are instances of the MTSCS problem that are efficiently solvable and others which are not. For the any-angle path planning problem and square lattice described above, we show that the MTSCS problem can be efficiently solved using a simple approach of iteratively adding paths to a control set if and only if they are not yet covered.

In this thesis, we provide a proof that the MTSCS problem is, in general, NP-complete. Motivated by this result, we develop a Mixed Integer Linear Programming or Mixed Integer Linear Program (MILP) encoding of the problem which can be solved using existing software like Gurobi [40]. This approach constitutes the only known non brute-force approach to solving the MTSCS problem. Though our approach does not scale with the size of the lattice, it is observed that a control set must be computed only once, offline.

Combining the two stages of lattice-based motion planning: Lattice trajectory generation and control set selection, results in a control set comprised of motion primitives that take into account the preferences of a user, and that optimizes a trade-off between the quality of compound maneuvers and the time required to compute them. This complete process is outlined in Figure 1.3. From a set of demonstrations, a set of weights that best describes these demonstrations is determined and used as input (along with the vertices of a lattice) to the proposed trajectory generation method. From here, a set of trajectories

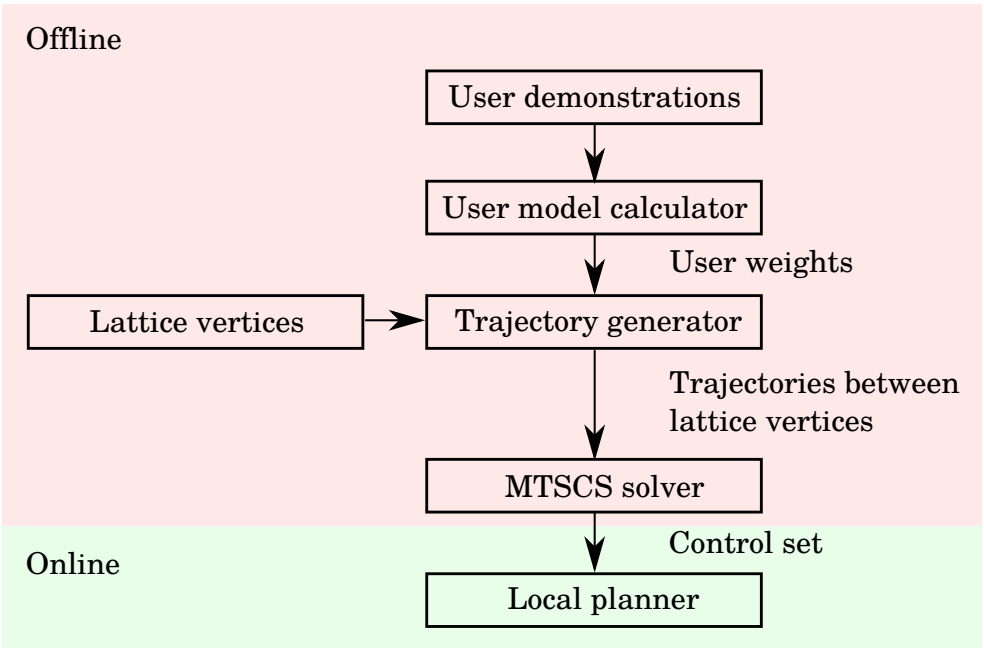


Figure 1.3: Block diagram illustrating how topics covered in this thesis provide an input control set for a local planner.

is developed between lattice vertices and a control set is selected. This control set is then used by a local planner to compute compound motions that do not collide with obstacles. In the next section, the structure of this thesis as well as the specific contributions of each chapter to the state of the art is described.

1.1 Thesis Structure & Summary of Contributions

The purpose of this section is to present an outline for the thesis. It will also highlight the contributions made to the state of the art in each chapter.

1.1.1 Chapter 2: Preliminaries

In this chapter, preliminary information for all the subsequent chapters is provided. The motion planning problem and the MTSCS problem are stated formally in this chapter.

Further, terms used informally thus far like *trajectory, state, configuration*, etc. are formally defined.

1.1.2 Chapter 3: The MTSCS Problem for an Any-Angle Square Lattice

This chapter addresses the MTSCS problem for an any-angle path planning problem with a square lattice (e.g., Figure 1.2). Here, the cost of a trajectory between two lattice vertices is taken to be the length of the vector between these vertices. The contributions of this chapter described here.

1. An efficient algorithm to solve this instance of the MTSCS problem is presented, and it is shown that this algorithm terminates in finite time for all values $t > 1$ even for infinite lattices.
2. Bounds on the size of the control set generated using the proposed algorithm are developed. These bounds are functions only of t for the case of an infinite lattice, and of t and the size of the lattice otherwise.
3. An upper bound on the sub-optimality of compound paths computed using the proposed control set (in the presence of obstacles) is presented.

The methods of this chapter are compared against the state of the art control set generation technique from [94].

1.1.3 Chapter 4: Tunable Trajectory Planner Using G^3 Curves

The problem of generating a trajectory between start and goal configurations – in the absence of obstacles – that optimizes an adjustable trade-off between travel time and comfort is addressed in this chapter. This problem is an instance of infinite-dimensional non-convex optimization. The contributions of this chapter are summarized below.

1. Given a set of weights representing the relative importance of comfort and travel time for a user, a method of simplifying the resulting infinite-dimensional non-convex optimization problem to one of finite dimension is proposed. Leveraging this simplification, a method to iteratively compute and refine both path and velocity components of the trajectory is presented.

2. To use the method proposed here, a technique to compute paths between start-goal configurations is developed. These paths are adjusted via a single parameter.

3. Finally, a technique to compute a velocity profile for a given path and user weights is proposed.

The methods in this chapter will be used in later chapters to compute trajectories between the vertices of a lattice. A control set for this lattice will then be selected by solving the MTSCS problem.

1.1.4 Chapter 5: High-Dimensional Lattice Planning with Optimal Motion Primitives

The MTSCS problem for a general lattice is discussed in this chapter. Further, applications of the problem for autonomous vehicles are developed. The work in this chapter is an extension of the authors' preliminary work in [12]. The specific contributions of this chapter are outlined here.

1. The NP-completeness of the decision version of the MTSCS problem for a general lattice is proved.

2. A mixed integer linear programming (MILP) formulation of the MTSCS problem for state lattices is proposed. This represents the first known non-brute force approach to solving the MTSCS problem.

3. An A*-based algorithm to compute feasible trajectories for difficult maneuvers in both parking lot and highway settings is developed. The algorithm accommodates off-lattice start and goal configurations up to a specified tolerance.

4. Finally, a novel algorithm that eliminates redundant vertices along trajectories computed using a lattice is introduced. This algorithm is based on shortest paths in directed acyclic graphs and runs in time quadratic in the number of motion primitives along the input trajectory.

1.1.5 Chapter 6: Learning Control Sets for Lattice Planners from User Preferences

This work investigates the problem of designing a motion planner that can capture user preferences. This problem involves simultaneously learning trajectories and a control set of fixed size given Demonstrations from a single user. These demonstrations are trajectories between configurations as driven by a user though we do not assume that users are able to perfectly demonstrate their preferred trajectories. This chapter presents a solution to this problem. The work presented in this chapter can be found in [13] and was a collaborative project with Nils Wilde, another PhD student at the University of Waterloo. Both authors contributed equally to this work.

1.2 Related Work

In this section, we present some of the work related to the topics of this thesis. For readability, we present the work related to each topic covered in this thesis separately.

1.2.1 Chapter 3: The MTSCS Problem for an Any-Angle Square Lattice

Any-angle trajectory planning has broad applications from video-games [75, 81, 94], to way-point computation for robotic navigation [65]. In [26, 72], the authors use any-angle trajectories as part of a heuristic for an A*-based graph search algorithm. It is shown in these papers that including any-angle trajectory costs from a start configuration to a goal in the presence of obstacles as part of a heuristic greatly reduces the run-time of A*.

Sampling-based techniques for any-angle planning include Asymptotically Optimal Rapidly-exploring Random Trees (RRT*) [54] and Fast Marching Tree (FMT*) [51] which produce trajectories that asymptotically converge to optimal with the number of samples. To improve the convergence rate of RRT*, the authors of [31] introduce Batch Informed Trees (BIT*) which draws samples from a decaying ellipse once an initial trajectory has been discovered. In [60], the authors present RRT-Connect, an algorithm which iteratively constructs, and attempts to connect, two trees – one rooted at the starting configuration and one at the goal – to decrease the time required to obtain an initial trajecotry.

In [43], the authors introduce ANYA, an algorithm that computes an optimal any-angle collision-free trajectory between start and goal configurations. This algorithm was later

optimized for run-time in [44]. Closely related to the work presented in Chapter 3, [75] illustrates the maximum error (relative to the optimal path in a lattice) incurred from using control sets of size 4,6, and 8 in both triangular and square lattices in 2 and 3 dimensions (in the absence of obstacles). In [21], an A*-variant called Theta* is introduced. When a vertex v is expanded by A* and a neighbor v' of v is determined, Theta* considers both v and the parent of v as potential parents of v'. The authors of [21] consider control sets of size 4, 8, and 16 noting the decreased sub-optimality of their computed paths (at the expense of computation time) as the size of the control set is increased. Increasing the control set size past 16 is considered in [94] in which the authors present an algorithm which takes as input a natural number $k \geq 2$ and returns a control set of size 2^k. The algorithm itself is based on the Fibonacci numbers, and the authors note that the sub-optimality of planned paths decreases as k increases (at the expense of computation time). In Chapter 3 it will be shown that identical sub-optimality can be obtained using control sets of much smaller size than those proposed in [94] by solving the MTSCS problem.

1.2.2 Chapter 4: Tunable Trajectory Planner Using G^3 Curves

As will be formalized in the following chapter, a trajectory is spatio-temporal construct comprised of a path in space and a velocity profile describing the traversal of that path through time. Related work on these topics is provided separately.

Path Planning

Path planning techniques can typically be divided into two categories: a path-fitting approach, and a curvature fitting approach. In the path fitting approach, a form of the path is fixed up to unknown parameters. These parameters are then computed numerically by solving optimization problems. Example path-forms include G^2-splines [87], and Bezier curves [41]. In the path fitting approach, smoothness and comfort of paths is typically enforced by minimizing the maximum curvature rate (the rate of change of curvature with respect to arc-length) over the trajectory. Minimizing the maximum curvature rate as opposed to simply bounding it may result in longer than necessary paths.

Unlike the path fitting approach, the curvature fitting approach assumes a form of the curvature profile over the path (as opposed to the path itself) up to unknown parameters. The unknown parameters are then computed by solving a two point boundary value problem between start and goal configurations. Early work in this approach includes Dubins'

paths [27] – shortest paths between start and goal configurations with continuously differentialble position with respect to arc-length and bounded curvature (i.e., G^1 paths). These paths, while efficient to compute, experience moments of infinite instantaneous jerk.

In [59, 112], the authors use a cubic polynomial curvature representation. However, the resulting path does not possess sufficient degrees of freedom to account for bounds on the curvature, resulting in potentially infeasible paths that violate physical curvature constraints of the vehicle.

Clothoid paths [29] address feasibility issues via constraints on curvature and curvature rate over the path resulting in paths with piece-wise constant curvature rate. A *clothoid*, or Euler Spiral, is a G^2 curve whose instantaneous curvature, κ is a linear function of the curve's arc-length, s. When the start and goal configurations are given as an (x, y) pose, a heading θ, and a curvature κ, several methods have been proposed to generate the shortest path from start to goal as a sequence of clothoid and straight line segments [102, 103]. When a vehicle is moving at constant speed, jerk is experienced lateral to the trajectory and is proportional to the derivative of curvature. Therefore, if the vehicles' path can be described by a clothoid, then it will experience piece-wise constant jerk when travelling at constant speed. Hence, by constraining the curvature rate, clothoid paths bound the maximum squared jerk at constant speed rather than minimizing the integral of the squared jerk, a widely accepted metric for discomfort [66].

Requiring that the curvature be twice differentiable with respect to arc-length, and placing bounds on curvature, curvature rate, and second derivative of curvature results in G^3 paths whose integral of the squared jerk can be minimized. Paths that are G^3 continuous are to clothoid (G^2) paths, what clothoids were to Dubin's (G^1) paths. Thus G^3 paths are one step closer to paths with infinitely differentiable G^∞ curvatures – like spline and Bezier paths – while minimizing arc-length in the presence of explicit restrictions on curvature, and curvature rate. In [77], the authors obtain G^3 paths by concatenating cubic spline curves and straight lines. Though this approach results in G^3 paths, the spline segments of the path, which are infinitely differentiable, may result in longer than necessary arc-lengths as is the case with regular spline paths. In fact it will shown in 4 that G^3 paths such that every sub-path is also G^3 (and not G^k for $k > 3$) are the shortest paths for which the integral of the squared jerk can be minimized.

In [6], the authors introduce Continuous Curvature Rate (CCR), and Hybrid Curvature Rate (HCR) curves. These curves are G^3 curves in which the derivative of curvature with respect to arc-length, σ, is a piece-wise linear, continuous function of the arc-length s, and the second derivative of curvature with respect to arc-length, ρ, is piece-wise constant. Every sub-path of these paths is also G^3 (not G^k for $k > 3$). The authors of [6] compute

paths between start and goal states by combining CCR/HCR curves and straight lines.

By placing bounds on σ and ρ, the paths developed in [6] have the added benefit of bounding not only the magnitude of the jerk at constant speed, but the angular jerk (the time rate of change of angular acceleration), and the snap (the time derivative of jerk) at constant speed. These benefits make HCR/CCR curves particularly attractive for use in trajectory planning.

The techniques presented in [6] have four drawbacks that are addressed in 4. First, the authors of [6] focus on paths whose start and goal states have either 0 or maximum curvature. Though the symmetry resulting from this assumption greatly simplifies the mathematics of path planning, it limits the usefulness of the approach. Second, the authors of [6] use search techniques to join HCR/CCR curves with straight lines to produce a path. These techniques can be time consuming. Third, it is assumed in [6], that the maximum second derivative of curvature with respect to arc-length, $\rho_{\max}$, along a path is known. Because HCR/CCR curves are categorized by a piece-wise constant functions $\rho(s)$, the value of $\rho_{\max}$ dictates the slope – and therefore the magnitude at any arc-length – of the curvature rate. Observe then, that $\rho_{\max}$ should be functions of speed as, for example, a driver may turn the steering wheel very quickly in a parking lot, but not on a highway. Finally, it is assumed in [6], that the velocity of the vehicle is a positive or negative constant.

The major criticism of path planning using clothoids/HCR/CCR curves is the time required to solve their inherent TPBV problems [34]. However, for all of the reasons stated above, HCR/CCR curves are excellent candidates for use in lattice-based motion planning where lattice trajectories are pre-computed.

Velocity Planning

Once a candidate path has been developed, a velocity profile to describe the motion of the vehicle along that path must be computed. This velocity profile should take into account a trade-off between comfort and duration of travel. A common technique involves minimizing a cost function that is a weighted sum of undesirable features [66]. In [121, 122], the authors integrate the weighted sum of the squared offset of the trajectory from the center of the road, the squared error in velocity from a desired profile, the squared acceleration, the squared jerk, and the squared yaw rate. A similar technique is employed in [67] with the addition of a penalization on final arc-length. Because the goal of the work in 4 is to produce trajectories between lattice vertices (from which a control set will be selected) in the absence of obstacles, obstacles such as road boundaries, pedestrians, etc.

are not considered. Therefore, the cost function used in 4 is similar to that employed by [67, 121, 122], but penalizes travel time, acceleration, jerk, and yaw rate.

In [119], the authors compute a velocity profile along a fixed path by minimizing the travel time required to traverse the path. This is done by leveraging the results of [71, 109] in which it is shown that computing the speed that minimizes travel time along a fixed path is a convex optimization problem. While the authors of [119] ensure the safety of their proposed velocity profile by bounding the magnitude of the centripetal force, the comfort of the resulting trajectory is not considered when planning the velocity profile. The addition of comfort metrics – acceleration, jerk, and yaw rate – to the cost function results in a non-convex optimization problem.

In [11], the authors propose a technique to compute a velocity profile for a fixed path that optimizes an arbitrary cost function. The assumption made by the authors is that the velocity profile is given by a single polynomial equation of the arc-length. They then state an optimization problem whose cost is given by the cost function, and whose decision variables are the coefficients of the velocity polynomial. This optimization problem is then solved using sequential quadratic programming. The main drawback of this approach is inability of polynomial functions to approximate continuous functions defined in a piece-wise fashion that feature some constant pieces. These latter functions are common for autonomous vehicles where a maximum velocity is reached and maintained for a duration of time. A more common approach to velocity planning for a path is to adopt a continuous piece-wise quadratic velocity model with potentially discontinuous, piece-wise constant, bounded, longitudinal jerk [46, 95]. The velocity profile proposed in 4 is piece-wise cubic in arc-length with continuous, bounded linear longitudinal jerk. We employ this model for three reasons. First, continuous longitudinal jerk (instead of simply bounded longitudinal jerk) allows the effect of longitudinal jerk on discomfort to be minimized as opposed to simply bounded. This is similar to the difference between G^2 and G^3 paths discussed above. Second, observe that adding a velocity profile to a path induces a re-parametrization of that path from arc-length to time. Selecting thrice continuously differentiable velocity profiles ensures the trajectory is C^3 continuous. Finally, because quadratic polynomials are a special case of cubic polynomials, a familiar piece-wise quadratic velocity model can be obtained via the methods presented here if that is desired.

In the above references, velocity profiles are planned for a fixed path. That is, a path is computed and then a velocity profile is developed for that path. As discussed above, an approach that iteratively refines both path and velocity profile is presented in 4. The result is a trajectory that considers velocity during the planning of the a path. In [39], the authors propose such a technique to develop a reference trajectory, and also propose a MPC-based tracking alogithm. However, the reference paths generated are G^1, and

the techniques are limited to lane changes on a highway. In [113], the authors develop comfortable trajectories for use in highway driving by computing a finite set of clothoid-spline-like s-shaped swerve trajectories that terminate at a configuration parallel to the center-line of the road. The trajectory with the smallest integral of the squared jerk is then selected from this set. The shape of the trajectories in the set and the assumption that a center-line is known limits this techniques generalizability. More recently, in [9], the authors combine numerical optimization techniques with lattice-based motion planning to compute trajectories for autonomous vehicles in unstructured environments like parking lots. In this work, the authors rely on a user-specified set of maneuvers to generate their motion primitives. Further, because the lattice trajectories are numerically derived, there is no guarantee that they will posses the property every sub-path of the trajectorys' path will be G^3. Further, numerically derived trajectories may require substantial storage space once pre-computed.

1.2.3 Chapter 5: High-Dimensional Lattice Planning with Optimal Motion Primitives

Broadly, trajectory planning techniques fall into one of four categories [34]: sampling based planners, interpolating curve planners, numerical optimization approaches, and graph search based planners.

Sampling based planners work by randomly sampling configurations in a configuration space and checking connectivity to previous samples. Two common examples include Asymptotically Optimal Rapidly-exploring Random Trees (RRT*) [54], and Probabilistic Roadmap (PRM) [58]. These methods make use of a local planner to expand or re-wire a tree to include new samples. Therefore, many TPBV problems must be solved online necessitating the use of simple kinematic models [70]. Thus feasibility may not be guaranteed [34]. In [4], a solution to this problem is proposed by replacing the the steering function which solves TPBV problems in RRT* with a simulation forward in time given a reference trajectory.

Sampling-based planners, while not complete, can be *asymptotically complete* with a convergence rate that worsens as the dimensionality of the problem is increased [104]. The performance of RRT* in higher dimensions is improved in [86] where the authors pre-compute a set of reachable configurations from which random samples are drawn. The reachable set is, in essence, a lattice and the approach in [86] is equivalent to repeatedly computing a single-element control set via random sampling of the lattice.

In [17], the authors state that the usefulness of sampling based techniques is considered

to be restricted to unconstrained motion planning problems, limiting their use in scenarios like highway driving.

Techniques using the interpolating curve approach include fitting Bezier curves [117], Clothoid curves [30], or polynomial splines [10] to a sequence of way-points. A typical criticism of clothoid-based interpolation techniques is the time required to solve TPBV problems involving Fresnel integrals [34, 93]. While TPBV problems are typically easily solved in the case of polynomial spline and Bezier curve interpolation, it is often difficult to impose constraints like bounded curvature on these curves owing to their low malleability [34, 93]. In [100], the authors develop a sampling-based planner that uses a control-affine dynamic model to facilitate solving TPBV problems. These trajectories are then smoothed via numeric optimization in [120] in which trajectories with non-affine non-holonomic constraints and piece-wise linear velocity profiles are developed. This work illustrates the benefits of computing way-points using a system of similar complexity to the desired final trajectory. However, by the very nature of the simplification, it is possible for infeasible rough paths to be developed. This is especially true when non-holonomic and obstacle constraints are considered as terms in a cost function for a numeric optimizer instead of as hard constraints (e.g., [25]). A common critique of both Interpolating curve and numerical optimization approaches is their reliance on global way-points [34, 93]. Moreover, interpolating curve techniques often face malleability and computation time issues [34, 93].

Lattice-based planners are versatile in the problems they address from motion planning in autonomous driving [9, 108] to manipulator robots [19], to UAVs [24, 82]. The authors of [73, 97] demonstrate lattice adaptations made to account for the structured environments of urban roads for use in autonomous driving, while in [24], a set of motion primitives is computed (based on experience) for a UAV exploring mine-shafts. A method of incorporating known way-points along a trajectory to a lattice-based motion planning framework is proposed in [107] resulting in a significant decrease in planning time. In [25, 98], the authors use motion primitives to traverse a lattice with states given by position and heading. In both these works, the authors use a rounding technique to account for off-lattice primitive concatenations due to non-cardinal headings. Compound trajectories comprised of several primitives are discontinuous in curvature a known source of slip and discomfort [66]. To alleviate this curvature discontinuity, the authors of [118] use the techniques of [25] to compute an initial trajectory which is then smoothed via numerical optimization. This approach requires at least as much computation time as the methods proposed in [25].

The authors of [119] compute motion primitives that minimize the integral of the squared jerk over the trajectory but limit their results to forward motion. In [76] the

authors consider the control set to be the entirety of the lattice. They define and search a graph whose vertices are the vertices of the lattice, and whose edges are all pairs of lattice vertices for which there exists a feasible, collision-free trajectory from the first vertex to the second. Adopting such a large control set may necessitate courser lattices with larger error with respect to free-space optimal.

The choice of motion primitives is of particular interest in this work. Typically motion primitives are chosen to achieve certain objectives for the paths they generate. For example, in [84], the notion of probabilistic motion primitives is introduced which achieve a blending of deterministic motion primitives to better simulate real user behavior. The authors of [52] use Dispertio, a dispersion minimizing algorithm from [80] to compute a set of motion primitives that result in trajectories with minimum dispersion. In [8], a set of motion primitives is computed that relies on a set of user-specified maneuvers. This work is then used in [9] to compute complex trajectories in unstructured environments. In [88], the authors present the notion of using a MTSCS of motion primitives similar to graph $t-$spanners first proposed in [85].

In [89], a heuristic for the MTSCS problem is presented which, though computationally efficient, does not have any known sub-optimality factor guarantees. Since a control set may be computed once, offline, and used over many motion planning problems, the time required to compute this control set is arguably of less importance than its size.

1.2.4 Chapter 6: Learning Control Sets for Lattice Planners from User Preferences

Closely related to the work in Chapter 6, [1] and [35] learn a human driving style from demonstrations, using a linear cost function weighting pre-defined features, and then generating paths using a graph or lattice. In contrast to the approach presented in this thesis, both earlier approaches do not compute a new lattice for a learned user cost function, but instead search over a given graph or lattice, using the updated cost function. Furthermore, they consider local features that describe the relation of the vehicle to the environment, together with global features that are independent of the environment. In contrast, this work only considers global features describing the driving style, i.e., the trade-off between travel time and passenger comfort, given a situation. Instead of keeping sub-optimal trajectories between lattice vertices, the proposed method recomputes all trajectories given a learned preference and then computes a sparse control set for the lattice. Thus, during the motion planning, the control set contains only trajectories that are optimal for the learned user preferences.

Research in human robot interaction studies how the behaviour of autonomous robots can be shaped to satisfy the preferences of users. Similar to work in learning from demonstration [2] or active preference learning [79, 99], a user cost function that puts weights on a set of features is employed in this work. Given a fixed set of features, the user's cost function is learned by estimating the weights on the features. In [37] tunable parameters are introduced to a motion planner suited for urban driving while the work of [15] proposes a set of features to identify different driving behaviours of human drivers.

Chapter 2

Preliminaries

The purpose of this chapter is provide background information on topics relevant to this thesis. In particular, formal definitions of terms hitherto used informally will be presented. We will also formally state the *motion planning problem* and the Minimum t-Spanning Control Set (MTSCS) problem.

2.1 Graph Theory

Following [61], a graph is an ordered pair $G = (V, E)$, where V is a set of vertices and E is a set of edges. In a weighted graph $G = (V, E, c)$ a real valued function associates a cost to each edge of the graph: $c : E \to \mathbb{R}$. We define a path $P_{s,g}$ in the graph between two vertices s and g in V as a sequence of edges $P_{s,g} = (e_0, \ldots, e_k)$ where $e_i = (v_i, v_{i+1})$, $v_0 = s, v_{k+1} = g$, and $v_j \neq v_m$ for all $j, m \in \{0, \ldots, k+1\}$. The cost of a path is defined as $c(P_{s,g}) = \sum_{e \in P_{s,g}} c(e)$. Observe that the path $P_{s,g}$ can also be uniquely defined as the sequence of vertices $P_{s,g} = (v_1, \ldots, v_{k+1})$ given above.

There are many known NP-complete problems involving graphs. One that is used in this paper is the *vertex cover problem* [55]. For an undirected graph $G = (V, E)$, a set $V' \subseteq V$ is called a *vertex cover* of G if for every edge $(i, j) \in E$, it holds that $(i \in V') \vee (j \in V')$.

Problem 2.1.1 (Vertex Cover)**.** Given an undirected graph $G = (V, E)$ and a natural number $K \geq 0$, determine if there exists a vertex cover V' of G with $|V'| \leq K$

A weighted graph $G = (V, E, c)$ is called *directed* if edges in E are ordered pairs. A weighted directed graph is called *acyclic*, if it contains no cycles. That is, for any vertex

$v \in V$ it is not possible to construct a non-trivial path $P_{v,v}$. Closely related to the notion of directed acyclic graphs is the notion of an arborescence:

Definition 2.1.2 (Arborescence)**.** From Theorem 2.5 of [62, Section 2.2], a graph G with a vertex s is an arborescence rooted at s if every vertex in G is reachable from s, but deleting any edge in G destroys this property.

In essence, a graph G is an arborescence rooted at s if for each vertex v in G other than s, there is a unique path in G from s to v. Arborescences are to directed graphs what *trees* are to undirected graphs.

2.1.1 Shortest Path in a Weighted Graph

Given a weighted directed graph $G = (V, E, c)$ as well as a pair of vertices $s, g \in V$, this section discusses algorithms that compute paths $P_{s,g}$ of minimal cost $c(P_{s,g})$. In particular, we present the two algorithms that are used in this work.

A* Shortest Path Algorithm: The A* algorithm, introduced in [45], computes the minimal cost path in a weighted directed graph $G = (V, E, c)$ from a starting vertex $s \in V$ to a goal vertex $g \in V$. The algorithm – summarized in Algorithm 1 – works by maintaining a lowest cost arborescence rooted at s and expanding that arborescence one edge at a time until the goal vertex is reached. This is accomplished by maintaining two sets: a set *Open* containing vertices that have been discovered by A* but whose edges have not yet been expanded, and a set *Closed* of vertices that have been discovered and whose edges have been expanded. The vertices of the arborescence is given by Open $\cup$ Closed. In each iteration of the while loop in Line 7, a vertex *current* that minimizes a cost given by the value fScore over all vertices in Open is selected (Line 8). In detail, for a vertex $v \in$ Open, fScore[v] is the sum of two values, gScore[v] and $h(v)$ where gScore[v] is the minimum cost path from s to v in the arborescence (and therefore in the graph), and $h(v)$ is an estimate of the minimum-cost path in G from v to g. Thus fScore[v] represents an estimate of the cost of the minimum-cost path from s to g in G that passes through v. Once selected, the neighbors of *current* are expanded (Line 15). If a neighbor v of *current* is not in Open, or if it is in Open but can be reached with lower cost via the vertex *current* (Line 19), then the gScore and fScore of v are updated (Lines 20-24). A* terminates once the goal g is the value with the lowest fScore in Open (Line 9), or if there are no more vertices in Open (Line 27). If g is the vertex in Open with the lowest fScore, then a path is constructed by backwards propagation through the arborescence (Lines 10-13).

The function $h : V \to \mathbb{R}_{\geq 0}$ used in computing the fScore of each vertex is called a *heuristic*. The value $h(v)$ represents an estimate of the cost of the minimal-cost path in G from v to g. A heuristic is called *admissible*, if it produces an under-estimate of the cost of the minimal-cost path in G from each vertex $v \in V$ to g. It is called *consistent* if for all vertices v, u with $(v, u) \in E$, it holds that $h(v) \leq c((v, u)) + h(u)$. It was shown in [45] that the A* algorithm presented in Algorithm 1 will always return the minimal-cost path in G from s to g if one exists (and it will return *Fail* otherwise) provided that h is *admissible* and *consistent*.

Algorithm 1 A* Graph-search algorithm

```
1:  procedure A*(G = (V, E), s, g, h)
2:      Open = {s}
3:      Closed = ∅
4:      cameFrom[s] =None
5:      gScore[s] = 0
6:      fScore[s] = h(s)
7:      while Open ≠ ∅ do
8:          current = arg min_{v∈Open} fScore[v]
9:          if current = g then
10:             path = [current]
11:             while cameFrom[current] ≠ None do
12:                 current = cameFrom[current]
13:                 add current to path
14:             return path in reverse order
15:         for all v ∈ V, (current, v) ∈ E do
16:             if v ∈ Closed then
17:                 continue
18:             temp = gScore[current] + c(current, v)
19:             if v ∉ Open or gScore[v] > temp then
20:                 cameFrom[v] = current
21:                 gScore[v] = temp
22:                 fScore[v] = gScore[v] + h(v)
23:                 if v ∉ Open then
24:                     add v to Open
25:         remove current from Open
26:         add current to Closed
27:     return Fail
```

In this work, we use heuristics $h(v)$ such that $h(v) \leq c((v,u)) + h(u)$ for all $v, u \in V$ thus guaranteeing consistency. Their admissibility will be discussed as they are introduced.

Shortest Path in a Directed Acyclic Graph: This section follows [20, Section 24.5, Pages 592-595]. Given a weighted, directed, acyclic graph $G = (V, E, c)$, a *topological ordering* of G is an ordering of the vertices of V such that if $(u, v) \in E$, then u appears before v in the ordering. Using this definition, a shortest path algorithm is summarized in Algorithm 2. The algorithm begins my constructing a topological ordering V' of V (Line 2), and initializing the current best cost to get (*cost*) from s to each vertex $v \in V'$ as well as the current predecessor (*cameFrom*) of each vertex $v \in V'$ (Lines 3-6). Next, each vertex in the topological ordering (Line 7) is expanded (Line 8) and the cost and predecessor of each neighbor is updated (Lines 9-11) if that neighbor can be reached with lower cost than previously discovered (Line 9). Finally a path is returned by backwards propagation of the predecessors (Lines 12-17).

Algorithm 2 Shortest Paths in a Directed Acyclic Graph

```
1:  procedure DAG-Shortest-Paths(G = (V, E), s, g)
2:      V' = topological ordering of V beginning with s
3:      for all v ∈ V' do
4:          cameFrom[v] = None
5:          cost[v] = ∞
6:      cost[s] = 0
7:      for all v ∈ V' do
8:          for all u, (v, u) ∈ E do
9:              if cost[v]+c((v, u)) < cost[u] then
10:                 cameFrom[u] = v
11:                 cost[u] = cost[v]+c((v, u))
12:     current = g
13:     path = [current]
14:     while cameFrom[current] ≠ None do
15:         current = cameFrom[current]
16:         add current to path
17:     return path in reverse order
```

It is shown in [20, Section 24.5, Pages 592-595] that such an algorithm returns the cost-minimizing path in G from s to g provided that it exists.

2.2 Differential Geometry & The Motion Planning Problem

In differential geometry, a *planar curve* (or simply *curve*) is a k-times continuously differentiable function $C : \mathcal{U} \to \mathbb{R}^2$ for some $k \in \mathbb{N}_{\geq 0}$. Here, $\mathcal{U}$ is an interval of $\mathbb{R}$, and we say that C is a curve *parameterized* by the *parameter* $u \in \mathcal{U}$. In this thesis, we assume that curves are k-times continuously differentiable, *and* $k+1$-times differentiable almost everywhere. This is a safe assumption for autonomous vehicle motion planning. Indeed, functions that are $k-$times continuously differentiable but *not* $k+1$-times differentiable almost everywhere are functions whose k^{th} derivative is continuous but almost never differentiable. Examples of such functions include the Weierstrass Functions illustrated in Figure 2.1 whose usefulness in motion planning is dubious.

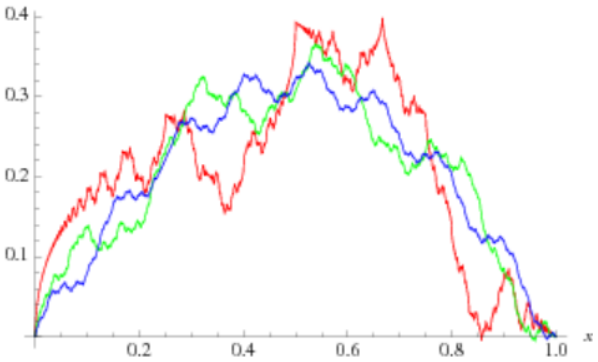


Figure 2.1: The Weierstrass Function, image courtesy of Wolfram Mathworld, available at https://mathworld.wolfram.com/WeierstrassFunction.html.

A *kinematic model* of a curve C is a vector of differential equations describing the changes in C that arise from changes in the parameter u [115, Chapter 1]. These differential equations must be sufficient to compute all $k+1$ derivatives of C with respect to u. For example, if the chosen parameter is arc-length, s, then a line-and-circle curve with bounded curvature (i.e., a $(k=1)$-times continuously differentiable curve comprised of straight lines and circular arcs with bounded radius connected continuously) may be described via the kinematic model

$$\begin{bmatrix} dx(s)/ds \\ dy(s)/ds \\ d\theta(s)/ds \end{bmatrix} = \begin{bmatrix} \cos(\theta(s)) \\ \sin(\theta(s)) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \kappa(s) \end{bmatrix} \tag{2.1}$$

where $(x, y) \in \mathbb{R}^2$ denotes position, θ heading, and κ curvature. Here, the curvature is

derived as follows: for a curve C, parameterized by arc-length, the tangent vector is given by

$$\boldsymbol{T}(s) = (dx(s)/ds, dy(s)/ds),$$

and we observe that $||\boldsymbol{T}(s)|| = \sqrt{(dx(s)/ds)^2 + (dy(s)/ds)^2} = \sqrt{\cos(\theta(s))^2 + \sin(\theta(s))^2} = 1$. Curvature is defined as the derivative of the unit tangent vector with respect to arc-length. Therefore,

$$\begin{aligned} |\kappa(s)| &= \left|\left|\frac{\boldsymbol{T}(s)}{ds}\right|\right| = \sqrt{\left(\frac{d^2x(s)}{ds^2}\right)^2 + \left(\frac{d^2y(s)}{ds^2}\right)^2} \\ &= \sqrt{\left(-\sin(\theta(s))\frac{d\theta(s)}{ds}\right)^2 + \left(\cos(\theta(s))\frac{d\theta(s)}{ds}\right)^2} = \left|\frac{d\theta(s)}{ds}\right|. \end{aligned}$$

Further, the sign of the curvature is taken to be the sign of $d\theta(s)/ds$. That is, curvature is positive for increasing heading and negative otherwise. Therefore, $d\theta(s)/ds = \kappa(s)$. In the kinematic model (2.1), $\kappa(s)$ may be discontinuous but is bounded for line-and-circle curves.

Elements that appear as outputs of a kinematic model are called *states*. In the example kinematic model (2.1), the states of the curve are given by x, y, θ while κ can be seen as an input to the kinematic model. For a curve C parameterized by $u \in \mathcal{U} = [u_0, u_1] \subseteq \mathbb{R}$, knowledge of the states at $u = u_0$, the inputs at all values $u \in \mathcal{U}$, and a kinematic model, is sufficient to determine the value of the states for all $u' \in \mathcal{U}, u' \geq u_0$ [18, Section 1.3]. A tuple of all of the states and inputs of a curve is called a *configuration* [16, Section 3.1] (e.g., (x, y, θ, κ) for the kinematic model (2.1)), while the set of all possible configurations is called a *configuration space*, $\mathcal{X}$. Finally, for autonomous vehicles, a *workspace* $\mathcal{W}$ is a subset of $\mathbb{R}^2$ defining the set of allowable planar positions. The workspace may represent the obstacle-free space $\mathbb{R}^2 - \mathcal{X}_{\text{obs}}$ for a set of obstacles $\mathcal{X}_{\text{obs}}$, or may chosen without reference to a set of obstacles. As mentioned in Chapter 1, a lattice is a discretization of the configuration space. A workspace may be thought of as a bounding box for the lattice with or without obstacles. For example, if one wishes to plan trajectories for robots in a room, then one may think of the boundaries of the room as the workspace, or one may include the static obstacles within the room as part of that workspace.

Intuitively, the set of states considered by a motion planner should be the smallest set of from which all desired properties and constraints of the vehicles' motion can be derived. For example, if one wishes to design a curve for an autonomous vehicle in which which the integral of the squared curvature is penalized, or the curvature is bounded, then curvature should be included as a state in configurations of the configurations space.

Consider a curve C_s parameterized by arc-length $s \in \mathcal{S}$ where $\mathcal{S}$ is an interval of $\mathbb{R}$ with left endpoint 0. Let $\mathcal{X}_s$ the configuration space of C_s. In this thesis, a *path* π is a function $\pi : \mathcal{S} \to \mathcal{X}_s$ where $\pi(s)$ is the configuration of the curve C_s at arc-length s.

In this thesis, a path is a purely spatial construct. That is, the definition of a path does not include a description of how the path is traversed through time. Augmenting a path from a spatial construct to a spatial-temporal construct can be achieved by including a *velocity profile* for the path. For the purposes of this thesis, a velocity profile for a path π is an $m-$times continuously differentiable function $v : \mathcal{S} \to \mathbb{R}$ for some $m \in \mathbb{N}_{\geq 0}$, and we assume that $v(s)$ is $(m+1)$-times differentiable almost everywhere (similar to the assumption made for curves). Velocity v relates time t to arc-length s via the differential equation $ds/dt = |v|$. Given a velocity profile v for a path π, a re-parameterization of π from s to t may be achieved as follows: let $t = T(s)$ be the function relating the time t required to reach arc-length s along π given v. Then,

$$T : \mathcal{S} \to \mathcal{T} : \; T(s) = \int_0^s \frac{ds}{|v(s)|}.$$

Observing that the integrand above is positive, $T(s)$ is a bijective function of s implying the existence of an inverse function. Letting $S = T^{-1}$, we may obtain the arc-length s along a trajectory at a given time t via $s = S(t)$, and the time to reach a given arc-length via $t = T(s)$. Therefore, given the tuple $(\pi(s), v(s))$, we may re-parameterize π in terms of time to produce a new function $\pi(S(t))$ given in terms of time. In order to properly describe the curve $\pi(S(t))$, however, the kinematic model of $\pi(s)$ is augmented to include v as well as its derivatives. Further, because the kinematic model is augmented, the set of states, configurations, and configuration space must be updated accordingly. We let $\mathcal{X}_t$ denote the updated configuration space. The function $\Pi : \mathcal{T} \to \mathcal{X}_t$ is called a *trajectory*.

For example, consider the line-and-circle path whose kinematic model is given by equation (2.1), and suppose that a velocity profile v for this path is continuous and piece-wise linear. Then a kinematic model for the resulting trajectory could be given by

$$\begin{bmatrix} dx(t)/dt \\ dy(t)/dt \\ d\theta(t)/dt \\ dv(t)/dt \end{bmatrix} = \begin{bmatrix} v(S(t))\cos(\theta(S(t))) \\ v(S(t))\sin(\theta(S(t))) \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ v(S(t))\kappa(S(t)) \\ a(S(t)) \end{bmatrix},$$

where a is acceleration. Here, the states the are x, y, θ, v with inputs κ, a and each configuration in the configuration space $\mathcal{X}_t$ of the trajectory is a tuple of the form $(x, y, \theta, \kappa, v, a)$.

Note that a path may be viewed as a trajectory with unit speed. In particular, if $v(s) = 1, \forall s \in \mathcal{S}$, then the time and arc-length functions reduce to $T(s) = s, S(t) = t$

implying that arc-length and time are equal. In this case, $\mathcal{X}_s = \mathcal{X}_t$ and $\Pi(t) = \pi(s)$. Therefore, the word *trajectory* can be used as a blanket term for both paths and trajectories with non-unit velocity profiles.

In light of the above definitions, we now state the motion planning problem.

Problem 2.2.1 (Motion planning problem)**. Input:** A kinematic model $\mathcal{K}$, with configuration space $\mathcal{X}_t$, a set of obstacles $\mathcal{X}_{\text{obs}}$, a set of magnitude constraints $\mathcal{M}$ on states in the configuration space, start/goal configurations $p_s, p_g \in \mathcal{X} - \mathcal{X}_{\text{obs}}$, a workspace $\mathcal{W}$, and a cost function c from the set of all trajectories with kinemtatic model $\mathcal{K}$ to the positive reals. **Output:** A trajectory $\Pi : \mathcal{T} \to \mathcal{X}_t$ where $\mathcal{T} = [0, t_f]$ is an interval of $\mathbb{R}$, and Π has the following properties:

1. **Adherence to Boundary Constraints:** It holds that $\Pi(0) = p_s, \Pi(t_f) = p_g$.
2. **Feasibility:** The kinematic model of the trajectory Π is given by $\mathcal{K}$, and at all times $t \in [0, t_f]$, each state in the configuration $\Pi(t)$ adheres to the magnitude constraints $\mathcal{M}$. Further, at all times $t \in [0, t_f]$, $\Pi(t) \in \mathcal{X}_t - \mathcal{X}_{\text{obs}}$. That is, Π does not collide with obstacles in $\mathcal{X}_{\text{obs}}$. Finally, at all times $t \in [0, t_f]$, the x, y components of $\Pi(t)$ lie within the workspace $\mathcal{W}$.
3. **Optimality:** Π minimizes cost c over all feasible trajectories that adhere to the boundary constraints.

Assumption 2.2.2. In Chapters 5, 4, we assume that the cost function c which is an input to the motion planning problem 2.2.1 is *additive*. That is, we assume that if Π_1 is a trajectory from a configuration i to a configuration j, and Π_2 is a trajectory from configuration j to configuration k, then the cost of a trajectory that first traverses Π_1 and then immediately traverses Π_2 is given by $c(\Pi_1) + c(\Pi_2)$. Costs of this form include any linear combination of travel time, arc-length, smoothness, and integrals of the squared magnitude of jerk, acceleration, yaw rate, curvature, etc. Such costs are common in motion planning for autonomous vehicles [26, 49, 119, 121]. This assumption will be slightly relaxed in Chapter 4 to include cost functions involving a maximum.

Though the motion planning problem is, in general, intractable [3], it is often possible for solution trajectories between start-goal pairs to be isometrically transformed to produce solution trajectories between other start-goal pairs. In detail, given a configuration space $\mathcal{X}$, let $\mathcal{P}$ denote the set of solutions to Problem 2.2.1 between all start-goal configurations in $\mathcal{X}$ in the absence of obstacles where $\mathcal{W} = \mathbb{R}^2$. We define an equivalence relation $R \subseteq \mathcal{P}^2$ as follows: two trajectories $\Pi_1(t), \Pi_2(t)$ are related if there exists an orientation-preserving

mapping ψ in the special Euclidean group of rigid motions from $\Pi_1(t)$ to $\Pi_2(t)$. That is, $(\Pi_1(t), \Pi_2(t)) \in R$ if $\Pi_1(t)$ can be rotated and translated to obtain $\Pi_2(t)$.

The equivalence relation induces a partition of $\mathcal{P}$ into equivalence classes $[\Pi(t)]_R = \{\Pi'(t) : (\Pi(t), \Pi'(t)) \in R\}$. We define a *motion* p as one of these equivalence classes. We define the motion p *from* $i \in \mathcal{X}$ *to* $j \in \mathcal{X}$ – written $i \cdot p = j$ – as the trajectory $\Pi \in p$ that solves Problem 2.2.1 with initial and final configurations i, j, respectively in the absence of obstacles, and with $\mathcal{W} = \mathbb{R}^2$. We also say that p is the *motion associated with* the trajectory $\Pi(t)$ if $\Pi(t) \in p$. For example, consider a left turn made by a vehicle at position $(1, 1)$ and with heading 0 with some velocity profile. The same left turn can be made at position $(2, 2)$ with heading $\pi/4$ with the same velocity profile. By rotating and translating the reference frame, the motions of the two left turns are can be made identical.

The concept of motions is leveraged in lattice-based motion planning which we discuss in the next section. .

2.3 Lattice-Based Motion Planning & The MTSCS Problem

In lattice-based motion planning, the motion planning problem 2.2.1 is simplified by restricting the set of available trajectories. In detail, a configuration space $\mathcal{X}$ from Problem 2.2.1 is discretized into a regularly repeating grid called a *lattice* $L \subseteq \mathcal{X}$ whose configurations are called *vertices*. Lattices are selected so as to ensure that the x, y-components of all lattice vertices lie within a given workpsace $\mathcal{W}$. For a given lattice, motions p are computed from each $i \in L$ to each $j \in L$ by solving Problem 2.2.1 offline in the absence of obstacles, with for $p_s = i, p_g = j$. A subset of these motions, called a *control set* whose elements are called *motion primitives* is then selected which represents the set of available motions at each lattice vertex. We elaborate on this process in the following sections and introduce the MTSCS problem.

2.3.1 Starting Set for a Lattice

In this section, we introduce the notion of a *starting set* $\mathcal{O}$ for a lattice L which is a set of *generalized* lattice vertices. To fully define this notion, we begin with a motivating example. Consider a square lattice L for an any-angle path planning problem.

For such a motion planning problem, the configuration space is given by $\mathcal{X} = \mathbb{R}^2$ with states x, y and configurations (x, y) representing position in the plane. By translation of

trajectories, a motion from any $i \in L$ to any $j \in L$ is equal to a motion from the origin $(0,0)$ to $j - i \in L$. Therefore, the set $\mathcal{B}$ of motions between all vertices $i, j \in L - (0,0)$ is equal to the set of motions between $(0,0)$ and all $i \in L - (0,0)$, and a control set E of motion primitives can be represented as a set of motions from $(0,0)$ to some of the vertices of L. This is illustrated in Figure 2.2. Here, the control set (red motions) is represented as a set of motions originating at the origin but can be used to define the set of available motions at any other vertex in the lattice (e.g., at $i = (2,3)$).

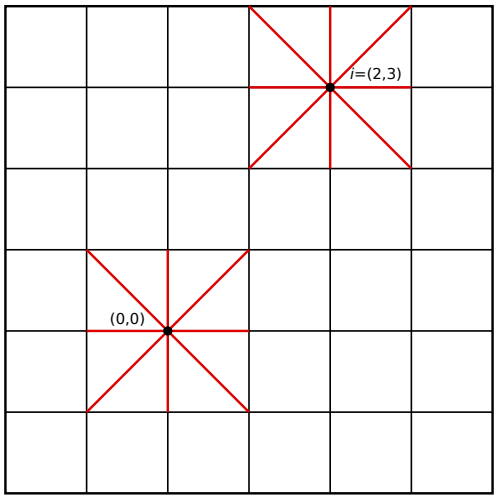


Figure 2.2: Example control set for an any-angle square lattice drawn at the origin.

For a lattice defined in the previous example, we consider the origin to be a *generalized* lattice vertex because defining a set of available motions at the origin (i.e., a control set) is sufficient to define the set of available motions at any other vertex in the lattice. Further, for each $i \in L$ and each $p \in E$, it holds that $i \cdot p \in L$. This last guarantees that motions in the control set take lattice vertices to lattice vertices.

For lattices that are more complex than the any-angle example given above, the origin may fail to generalize all other vertices. This can arise if the configuration space contains states other than position and heading, or if a motion from $i \in L$ to $j \in L$ would result in an off-lattice configuration when initiated at the origin. For example, consider configurations in a configuration space of the form (x, y, θ, κ) for position $(x, y) \in \mathbb{R}^2$, heading $\theta \in [0, 2\pi)$, and curvature $\kappa \in \mathbb{R}$. Let $i = (x_i, y_i, \theta_i, \kappa_i), j = (x_j, y_j, \theta_j, \kappa_j)$ and let p be a motion from i to j. If $\kappa_i \neq 0$, then p will not be a motion from the origin to any configuration in $\mathcal{X}$.

The above example illustrates how the origin fails to generalize all lattice vertices if curvature is included as a state. However, this failure may persist even for configurations of

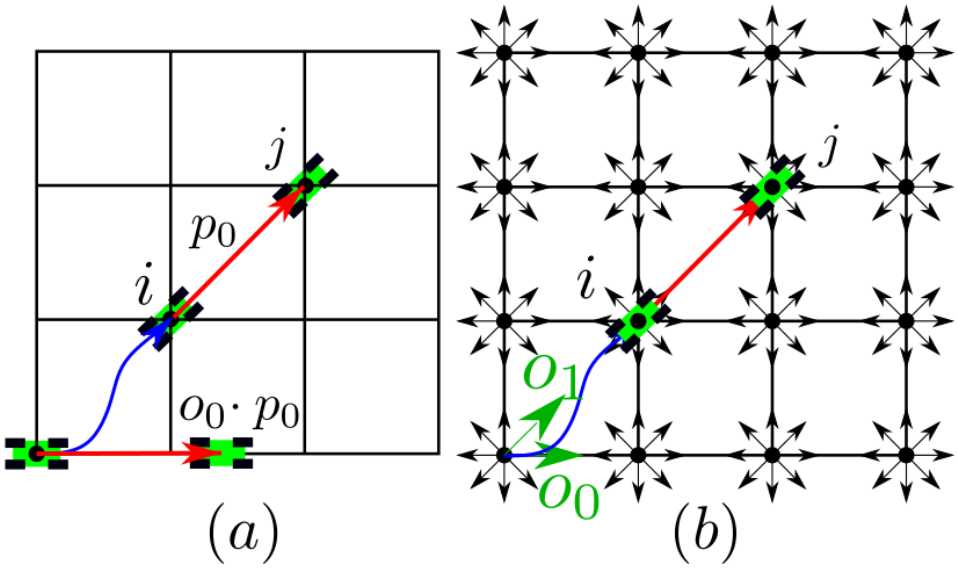


Figure 2.3: (a) Config. space, lattice and start from (2.2). (b) Lattice and $\mathcal{X}$ as in (a) with start set $\mathcal{O} = [o_0 = (0, 0, 0), o_1 = (0, 0, \pi/4)]$.

the form (x, y, θ) if a motion from $i \in L$ to $j \in L$ would result in an off-lattice configuration when initiated at the origin. For example, let

$$\begin{aligned} \mathcal{X} =& \mathbb{R}^2 \times [0, 2\pi], \ L = \mathbb{Z}^2 \times \{i\pi/4, i = 0, \ldots, 7\}, \\ o_0 =& (0, 0, 0), \ \mathcal{W} = \mathbb{R}^2, \end{aligned} \tag{2.2}$$

denote configurations space, lattice, origin, and workspace, respectively – as in Figure 2.3 (a). If $i = (1, 1, \pi/4) \in L$ and $j = (2, 2, \pi/4) \in L$, then the motion p_0 from i to j is such that $o_0 \cdot p_0 \notin L$. Therefore, the simple diagonal motion p_0 will not appear in any control set E defined as motions originating at the origin. This may result in motions with excessive oscillations.

The origin is not a generalized vertex for a lattice with states other than position and heading or off-lattice motions. However, by including several starting vertices in a *starting set* $\mathcal{O} \subset L$ ('O' for *origin*), we may increase the number of lattice vertices that are generalized by an $o \in \mathcal{O}$. Here, elements $o \in \mathcal{O}$ are called *starts*. The notion of a start *generalizing* a vertex is formalized now. Given a set of starts $\mathcal{O} \subset L$, vertices $i, j \in L$, a workspace $\mathcal{W}$, and motion p from i to j, we say that $i \cdot p$ is a *valid concatenation*, if

1. Vertex i is *generalized* by $\mathcal{O}$. That is, there exists a start $o \in \mathcal{O}$ such that $o \cdot p \in L$. Here, we say that o *generalizes* i.

2. Motion p starting at i is *contained within the workspace*. That is, if $\Pi(t)$ is the trajectory from i to j associated with the motion p then for all $t' \in [0, t_f]$, the x, y-components of $\Pi(t')$ lie within $\mathcal{W}$.

In the example in Figure 2.3 (b), let $\mathcal{O} = [o_0 = (0, 0, 0), o_1 = (0, 0, \pi/4)]$, let p_0 be the motion from o_0 to i, and let p_1 be the motion from o_1 to i. Then, observe that $i \cdot p_1 = j$ is a valid concatenation. Moreover, $o_0 \cdot p_0 \cdot p_1 = j$ is a string of valid concatenations from o_0 to j containing a diagonal motion. Motivated by the examples above, an ideal set of starts $\mathcal{O}$ would have the following property:

Property 2.3.1 (Property of an ideal set of starts)**.** For every pair of vertices $i, j \in L$ with motion p from i to j, if p starting at i is contained in $\mathcal{W}$, then concatenation $i \cdot p$ is valid. That is, there exists a start $o \in \mathcal{O}$ such that $o \cdot p \in L$ (i.e., o generalizes i).

Obtaining a set of starts with Property 2.3.1 is simple for a class of configuration spaces described here. For motion planning for autonomous cars and car-like robots, we typically use a configuration space of the form

$$\mathcal{X} = \mathbb{R}^2 \times [0, 2\pi) \times U_0 \cdots \times U_N, \tag{2.3}$$

for position $(x, y) \in \mathbb{R}^2$, heading $\theta \in [0, 2\pi)$, and $U_i \subset \mathbb{R}$ higher order states for $i = 0 \dots N$. These may include curvature, curvature rate, velocity, acceleration, jerk, etc. We make the assumption that each state U_i is bounded. These bounds may reflect the physical limitations of a vehicle, passenger comfort, or speed limits. This assumption allows us to write $U_i = [U_i^0, U_i^1] \subset \mathbb{R}$ for some upper and lower limits $U_i^0, U_i^1 \in \mathbb{R}$. For $\mathcal{X}$ in (2.3), we construct a lattice $L \subseteq \mathcal{X}$ by discretizing each state separately. In particular, for $\alpha, \beta \in \mathbb{R}_{>0}$, and $n_0, n_1, n_2, m_i \in \mathbb{N}$ for $i = 0 \dots N$, we let

$$\begin{aligned} L = &\{i\alpha, i = -n_0 \dots n_0\} \\ &\times \{i\beta, i = -n_1 \dots n_1\} \\ &\times \{\pi i / 2^{n_2 - 1}, i = 0 \dots 2^{n_2} - 1\} \\ &\times \prod_{i=0}^{N} \{U_i^0 + j/m_i, j = 0 \dots (U_i^1 - U_i^0) m_i\}. \end{aligned} \tag{2.4}$$

This lattice samples $2n_0 + 1$, and $2n_1 + 1$ values of the x, y coordinates, respectively, with spacing α, β between samples, respectively. It partitions heading values $[0, 2\pi)$ into 2^{n_2} evenly spaced samples, and for $i = 0 \dots N$, it partitions U_i into m_i evenly spaced samples.

For $\mathcal{X}, L$ from (2.3), (2.4), let

$$\begin{aligned}\mathcal{O} = \Big\{ & (0,0,\theta,u_0,\dots u_N) \ : \\ & \theta \in \{j\pi/2^{n_2-1}, j=0,\dots,2^{n_2-2}-1\}, \\ & u_i \in \{U_i^0 + j/m_i, j = 0 \dots (U_i^1 - U_i^0)m_i\}\Big\}.\end{aligned} \tag{2.5}$$

This starting set has Property 2.3.1. An example of a size-4 starting set $\mathcal{O}$ for a lattice with vertices of the form (x, y, θ) with headings $\pi i/8, i = 0, \dots 15$ and $\mathcal{O}$ given by (2.5) is illustrated in Figure 2.4. These motions were computed by solving the MTSCS problem described later.

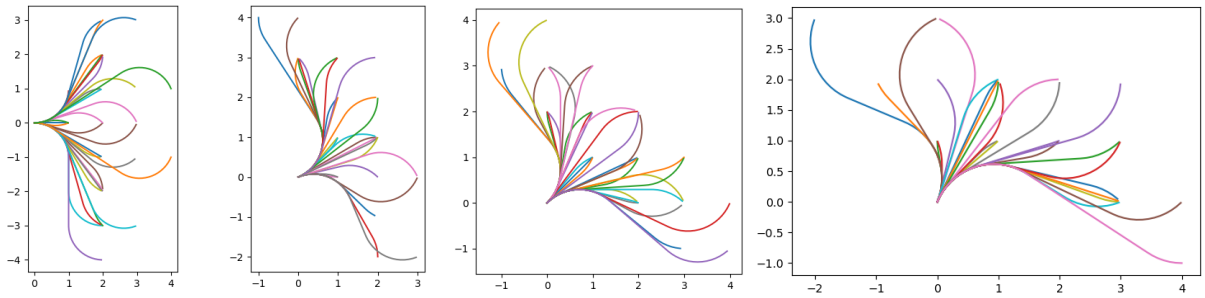


Figure 2.4: Control set $E = \bigcup_{o\in\mathcal{O}} E_o$ for $\mathcal{O} = \{(0,0,0), (0,0,\pi/8), (0,0,,\pi/4), (0,0,3\pi/8)\}$ for a lattice with 16 headings. Motions were planned using Dubins' paths.

In the next section we detail how a set of starts $\mathcal{O}$ with this property is used in lattice-based motion planning. We also introduce the MTSCS problem.

2.3.2 Lattice-Based Motion Planning with Starting Sets

In this section, we describe how a set of starts (which may or may not have Property 2.3.1) can be used in motion planning and we formulate the MTSCS problem. In this work, we assume that for all $i, j \in \mathcal{X}$, there exists a trajectory $\Pi(t)$ with associated motion p solving Problem 2.2.1 from i to j in the absence of obstacles if $\mathcal{W} = \mathbb{R}^2$. We assume further that the cost of this motion, $c(p)$ is known, non-negative, obeys the triangle inequality, and is equal to 0 if and only if $i = j$. Here, the cost $c(p)$ is the cost of the trajectory $\Pi(t)$ that solves the motion planning Problem 2.2.1 for $p_s = i, p_g = j$ in the absence of obstacles and with $\mathcal{W} = \mathbb{R}^2$. This cost is left general up to Assumption 2.2.2, and may represent arc-length, travel time, integral of the squared jerk, etc.

Given the tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W})$ of configurations space, lattice, starting set (with or without Property 2.3.1), cost of motions, and workspace, respectively, the high level idea behind motion planning using a starting set is the following. For each $o \in \mathcal{O}$, we pre-compute sets

$$\mathcal{B}_o = \{p : o \cdot p \in L - \mathcal{O}\},\ \mathcal{B} = \bigcup_{o \in \mathcal{O}} \mathcal{B}_o, \tag{2.6}$$

of motions from o to each vertex $i \in L - \mathcal{O}$, and the union of those motions over all starts $o \in \mathcal{O}$, respectively. Critically, we make the following assumption:

Assumption 2.3.2. We assume that each motion $p \in \mathcal{B}$ is valid. Since motions in $\mathcal{B}$ start at starting vertices $o \in \mathcal{O}$, this assumption is equivalent to assuming that these motions when starting at o are contained in $\mathcal{W}$. That is, the direct motion from each $o \in \mathcal{O}$ to each vertex in $L - \mathcal{O}$ is contained in $\mathcal{W}$.

Observe that if $\mathcal{O}$ has Property 2.3.1 then by Assumption 2.3.2, for any pair of vertices $(i, j) \in L \times (L - \mathcal{O})$ if the motion p from i to j is contained in $\mathcal{W}$ when starting at i, then $p \in \mathcal{B}$.

Once $\mathcal{B}$ has been computed, we – offline – select a control set $E \subseteq \mathcal{B}$. In particular, for each $o \in \mathcal{O}$, we choose a subset $E_o \subseteq \mathcal{B}_o$ of motions form o to vertices in $L - \mathcal{O}$. The set E will be used to determine the set of available motions at a lattice vertex. To this end, we offer the following definition.

Definition 2.3.3 (Relative start)**.** For vertex $i \in L$, the *relative start* of i is the starting vertex $R(i) \in \mathcal{O}$ that generalizes i for all motions p from i to vertices in $L - \mathcal{O}$ (if it exists). That is, $R(i)$ is the relative start of i if for all $j \in L - \mathcal{O}$ with motion p from i to j, $R(i) \cdot p \in L$.

Intuitively, the relative start of $i \in L$, is the vertex $o \in \mathcal{O}$ that is *most like* i. It is in fact, the vertex in $R(i) \in \mathcal{O}$ such that any trajectory starting at $R(i)$ is in the same equivalence class p as trajectories starting at i. In the example presented in Figure 2.3 (b), the relative start of i is $R(i) = o_1$ and if p_0 is the motion from i to j (in this case the diagonal red motion), then there exists a $j' \in L$ (in this case $j' = i$) such that p_0 is the motion from $R(i)$ to j'.

If $L, \mathcal{O}$ are given by (2.4), (2.5), respectively, then the relative start of any configuration $i = (x, y, \theta, u_1, \ldots, u_N) \in L$ is $R(i) = (0, 0, \theta', u_1, \ldots, u_N)$ where

$$\theta' = \frac{\pi}{2^{n_2-1}} \left(\left(\frac{2^{n_2-1}\theta}{\pi} \right) \mod 4 \right).$$

Using the notion of relative starts, we reduce the problem of computing a motion between lattice vertices to computing the shortest path in a graph. Given a control set E, let

$$\begin{aligned}\bar{E} &= \{(i,j) \in L^2 : (i \cdot p = j \text{ is valid}) \wedge \big(p \in E_{R(i)}\big)\},\\ \bar{E}_{\text{CF}} &= \{(i,j) \in E : (i \cdot p = j) \wedge (p \text{ does not collide with an obstacle})\}.\end{aligned} \tag{2.7}$$

Intuitively, $\bar{E}$ – referred to as a *connection set* – is the set of all tuples $i, j \in L$ such that 1) a motion from i to j is also a motion from $R(i) \in \mathcal{O}$ to a vertex in $L - \mathcal{O}$ (in the absence of obstacles), and 2) the motion p from i to j is contained in $\mathcal{W}$ when starting at i. The set $\bar{E}_{\text{CF}}$ – called the collision-free connection set – is the set of elements $(i,j) \in \bar{E}$ such that the motion from i to j does not collide with an obstacle. Here, CF stands for *collision-free*. Computing a motion between lattice vertices given a control set E is thus equivalent to computing a minimum-cost path in the weighted directed graph $G_{\text{CF}} = (L, \bar{E}_{\text{CF}}, c)$ with vertices L, edges given by the collision-free connection set $\bar{E}_{\text{CF}}$, and cost $c((i,j))$ equal to $c(p)$ where p is the motion from i to j. Similar to G_{CF}, we let $G_{\text{Free}} = (L, \bar{E}, c)$ be the graph G_{CF} in the absence of any obstacles (note that in the absence of obstacles, $\bar{E} = \bar{E}_{\text{CF}}$).

2.3.3 Selecting a Control Set: The MTSCS problem

Thus far, we have described how a set of starts $\mathcal{O}$ and a control set E can be used in motion planning. We now address how E can be computed beginning with two definitions.

Definition 2.3.4 (Path using E)**.** Given the tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W}, E)$ of configuration space, lattice, starting set, cost, workspace, and control set, respectively, the *path using* E from $o \in \mathcal{O}$ to $j \in L$, denoted $\pi^E(o,j)$ is the cost-minimizing path from o to j (ties broken arbitrarily) in the weighted, directed graph $G_{\text{Free}} = (L, \bar{E}, c)$.

Definition 2.3.5 (Distance using E)**.** Given the tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W}, E)$, the *distance using* E from $o \in \mathcal{O}$ to $j \in L$, denoted $d^E(o,j)$ is the cost of the path using E from o to j.

By Assumption 2.2.2, the distance using E from $o \in \mathcal{O}$ to $j \in L$ is the cost of the compound motion from o to j comprised of sub-motions in E. Let $i \in L - \mathcal{O}, o \in \mathcal{O}$. Observe that if $E = \mathcal{B}$, then $d^E(o,i) = d^{\mathcal{B}}(o,i) = c((o,i))$. That is, if $E = \mathcal{B}$ then the direct motion from o to i is an available motion at o by Assumption 2.3.2 and the distance using $E = \mathcal{B}$ from o to i is the cost of the trajectory that solves Problem 2.2.1 with $p_s = o, p_g = i$ in the absence of obstacles. This implies that using $\mathcal{B}$ as a control set will result in trajectories with minimal cost in the absence of obstacles. However, because $\mathcal{B}$ can be large, the branching factor at a vertex $i \in L$ (given by $|E_{R(i)}|$) during an online search

of the graph $G_{\text{CF}} = (L, \bar{E}_{\text{CF}} = \bar{\mathcal{B}}_{\text{CF}}, c)$ may be prohibitive. We therefore wish to limit the size of $|E_{R(i)}|$ in such a way that does not overly worsen $d^E(o, j)$ for all $j \in L - \mathcal{O}, o \in \mathcal{O}$. This motivates the following definition:

Definition 2.3.6 (t-Error)**.** Given the tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W}, E)$, the $t-$*error* of E is defined as

$$t\text{Er}(E) = \max_{\substack{o \in \mathcal{O} \\ j \in L - \mathcal{O}}} \frac{d^E(o, j)}{d^{\mathcal{B}}(o, j)}.$$

That is, the $t-$error of a control set E is the worst-case ratio of the distance using E from a start $o \in \mathcal{O}$ to a vertex $j \in L$ to the cost of the direct motion from s to j taken over all starts $o \in \mathcal{O}$ and vertices $j \in L$. The $t-$error is a metric for the *quality* of a control set. Intuitively, the $t-$error is a measure of the regret incurred by selecting E as a control set instead of $\mathcal{B}$.

Definition 2.3.7 ($t-$Spanning Set)**.** Given the tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W}, E)$, and a real number $t \geq 1$, we say that a set E is a $t-$*spanner* of L (or that E $t-$*spans* L), if $tEr(E) \leq t$.

Ideally, a control set would have both small $t-$error and small values of $|E_o|$ for all $o \in \mathcal{O}$. Suppose $t > 1$ and observe trivially that $E = \mathcal{B}$ will $t-$span L. This begs the question: is there a smaller control set E (i.e., a control set with $|E_o| < |\mathcal{B}_o|$ for all $o \in \mathcal{O}$) that will also $t-$span L? This question is formulated formally by the MTSCS problem [89]:

Problem 2.3.8 (Minimum $t-$spanning Control Set Problem)**. Input:** A tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W})$, and a real number $t \geq 1$.
Output: A control set $E = \bigcup_{o \in \mathcal{O}} E_o$ that $t-$spans L where $\max_{o \in \mathcal{O}} |E_o|$ is minimized.

Using a solution E to Problem 2.3.8 as a control set for lattice planning has two beneficial properties. First, the branching factor at any vertex $i \in L$ during an online search of the graph G_{CF} is given by $|E_{R(i)}|$ whose maximum value $\max_{o \in \mathcal{O}} |E_o|$ is minimized. Second, for every vertex $i \in L - \mathcal{O}$ and every $o \in \mathcal{O}$, it must hold that $d^E(o, i) \leq t d^{\mathcal{B}}(o, i) = tc((o, i))$. Thus, Problem 2.3.8 represents a trade-off between branching factor and the quality of computed motions. The decision version of Problem 2.3.8 is given here:

Problem 2.3.9 (MTSCS Problem – Decision Version)**.** Given the tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W})$, a real number $t \geq 1$, and a natural number $K \geq 0$, determine if there exists a control set $E = \bigcup_{o \in \mathcal{O}} E_o$ that $t-$spans L such that $\max_{o \in \mathcal{O}} |E_o| \leq K$.

Very similar to the MTSCS problem is the Minimum Spanning K-Control Set (MSKCS) problem. This problem – addressed in Chapter 6 – is extremely related to the MTSCS problem and, in fact, has an almost identical solution.

Problem 2.3.10 (MSKCS Problem)**. Input**: A tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W})$, and a natural number $K \geq 0$.
Output: A control set $E = \bigcup_{o \in \mathcal{O}} E_o$ such that $\max_{o \in \mathcal{O}} |E_o| \leq K$ and $t\text{Er}(E)$ is minimized.

In essence, instead of minimizing the control set size given an acceptable error (the MTSCS problem), the MSKCS problem minimizes the error given an acceptable control set size. The decision version of the MSKCS problem is:

Problem 2.3.11 (MSKCS Problem – Decision Version)**.** Given the tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W})$, a natural number $K \geq 0$ and a real number $T \geq 1$, determine if there exists a control set $E = \bigcup_{o \in \mathcal{O}} E_o$ such that $\max_{o \in O} |E_o| \leq K$ and $t\text{Er}(E) \leq T$.

Chapter 3

The MTSCS Problem for an Any-Angle Square Lattice

3.1 Introduction

In this chapter, we address the Minimum $t-$Spanning Control Set (MTSCS) problem 2.3.8 for a simple instance. In particular, we consider the problem of computing a MTSCS for the tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W}^n)$ with configuration space $\mathcal{X}$ comprised of configurations of the form (x, y), square lattice L, starting set $\mathcal{O}$ with a single element located at the origin, cost c of motions given by the Euclidean distance between the endpoints of those motions, and workspace $\mathcal{W}^n = [-n, n]^2$ for a number $n \in \mathbb{N}_{\geq 1} \cup \{\infty\}$. In this chapter, a motion p that solves the motion planning problem 2.2.1 between lattice vertices i, j in the absence of obstacles is a straight line connecting i and j, and the lattice is given by

$$L^n = \mathbb{Z}^2 \cap \mathcal{W}^n = \{(x, y) \in \mathbb{Z}^2 : x, y \in [-n, n]\}. \tag{3.1}$$

Because there is only one starting vertex $o = (0, 0)$, a control set E can be uniquely defined by a subset of the lattice vertices. That is, we may write $E = \{v_1, \ldots, v_r\}$ with the understanding that the *motion* associated with $v_i \in E$ is the straight line connecting o and v_i.

In this chapter, we will provide an efficient and intuitive algorithm that solves the MTSCS problem for the problem instance described above. We will also show that the MTSCS is finite even if $n = \infty$ provided that $t > 1$, and we provide bounds on its size. Further, we will provide a bound on the maximum cost error incurred from using

the proposed control set with respect to free-space optimal in the presence of obstacles. Finally, we compare our technique to a state-of-the-art control set generation method in any-angle path planning and show that the former results in exponentially smaller control sets than the latter for the same $t-$error (Defined in 2.3.6).

We begin this chapter with a motivating example. Consider a continuous space version of the MTSCS problem for the instance described above. That is, suppose we wish to compute a set $E = \{l_i, i = 1, \ldots, r\}$ of straight lines l_i passing through the origin and some configuration (x_i, y_i) – as in Figure 3.1 (a).

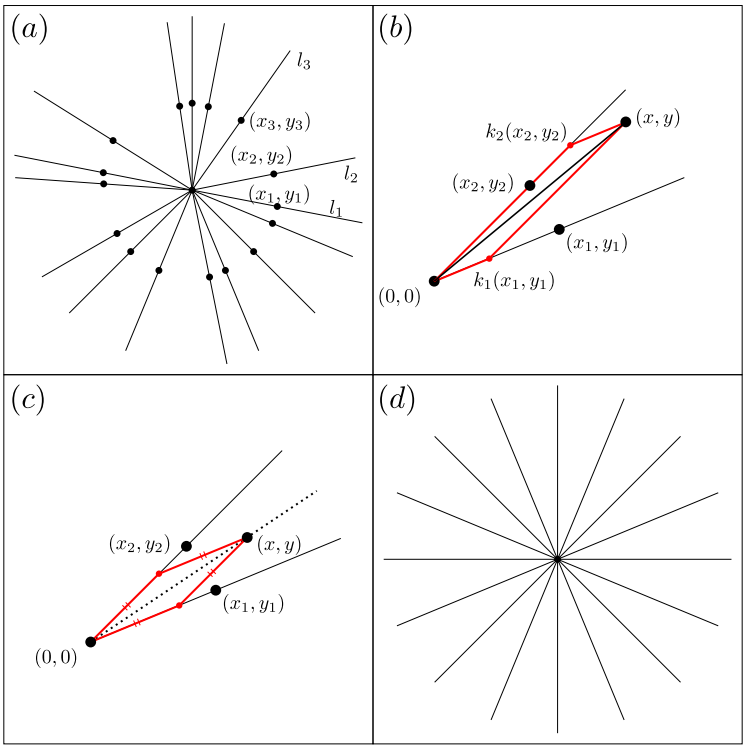


Figure 3.1: Motivating Example.

Observe that the set E partitions $\mathbb{R}^2$ into a set of *wedges* – sectors between lines $l_i, l_{i+1}, i = 1, \ldots, r-1$. Consider one such wedge w between, say, l_1 and l_2. Observe that for any $(x, y) \in w$ there will always exist non-negative real numbers k_1, k_2 such that $k_1(x_1, y_1) + k_2(x_2, y_2) = (x, y)$ provided $(x_1, y_1), (x_2, y_2)$ are linearly independent. This is illuatrated in Figure 3.1 (b). Suppose that we consider w to be *covered* by E when each

$(x, y) \in w$ is such that

$$\frac{k_1||x_1, y_1|| + k_2||x_2, y_2||}{||x, y||} \leq t, \text{ where}$$
$$(x, y) = k_1(x_1, y_1) + k_2(x_2, y_2), \tag{3.2}$$

for a given value $t \geq 1$ and where $||\cdot||$ represents the Euclidean norm. A continuous version of the MTSCS problem for the tuple $(X, L, \mathcal{O}, c)$ defined above is: compute a set E of straight lines passing through the origin such that every wedge w is covered by E given an input value t, and E is of minimal size.

It is not difficult to show that for each wedge w for a given set E, any configuration lying on the line that bisects w will maximizes the ratio in (3.2) (see Figure 3.1 (c)). Further, the ratio at these configurations is given by $2/\sqrt{2+2\cos(\theta)}$ where θ is the angle inscribed by the wedge.

Therefore an intuitive solution to the continuous MTSCS problem would be to continually bisect the plane until each wedge is *sufficiently narrow* so as to guarantee that $2/\sqrt{2+2\cos(\theta)} \leq t$ – as illustrated in Figure 3.1 (d). This is, in essence, a greedy solution: we construct a control set E recursively by adding lines along which the error ratio in (3.2) is maximized. From another perspective: we add lines to E that maximally reduce the angle θ in each in each resulting sub-wedge. This intuitive greedy approach is, in fact, optimal for the continuous MTSCS problem (the proof follows from the result of this chapter). However, the existence of efficient algorithms for continuous problems does not guarantee the existence of efficient algorithms for the discrete versions of those problems.

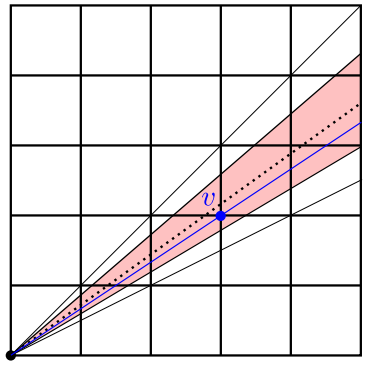


Figure 3.2: Modification of continuous algorithm for discrete lattices.

In this chapter, we show that the intuitive algorithm presented above also solves the MTSCS for the discrete lattice (3.1) with minor modifications. In particular, we construct

a control set E of vertices by selecting the shortest lattice vertex for which the inequality (3.2) fails in each wedge. This vertex v is the shortest lattice vertex such that the line passing through the origin and v is sufficiently close to the bisecting line of the wedge to ensure that the inequality in (3.2) fails. This is illustrated in Figure 3.2 where the red region about the bisecting line of the wedge represents the set of vertices for which the inequality in (3.2) fails, and v is the shortest such vertex.

In the next section, we present the algorithm that solves the MTSCS problem for the Lattice L^n for all $n \in \mathbb{N}_{\geq 1} \cup \{\infty\}$. The completeness of this algorithm is established in Section 3.4 after technical results in Section 3.3. .

3.2 Main Result

In this section we present the main result of this chapter: an algorithm that solves the MTSCS problem for the lattice L^n in (3.1). The approach is summarized in Algorithm 3.

Algorithm 3 Greedy MTSCS solver

1: **procedure** MTSCS($L^n, t > 1$)
2: $E = \{(1,0),(0,1),(-1,0),(0,-1)\}$
3: **for** $v \in L^n$ in ascending order of cost **do**
4: **if** v is not $t-$reachable using E **then**
5: $E = E \cup \{v\}$
6: Sort E in increasing order of $\arctan_2(y_i, x_i)$
7: **if** TestSet(E, L^n, t) **then**
8: **return** E

Algorithm 3 starts with a control set E given in Line 2. It then processes lattice vertices in order of length, adding them to E if and only if the ratio of the distance using E to v (defined in 2.3.5) to the cost of the direct motion from o to v – given by $||v||$ – exceeds t. In this case, we say that v is not $t-$*reachable*, meaning that v cannot be reached within a factor of t of $||v||$. Algorithm 3 terminates if the sub-formula `TestSet` returns *True*. We will show later that this last ocurrs if and only if every vertex in L^n is $t-$reachable using E. For now, we present the `TestSet` algorithm:

Algorithm 4 Stopping Criteria

```
1: procedure TestSet(E = {(x_1, y_1), (x_2, y_2), ..., (x_q, y_q)}, L^n, t)
2:     if n < ∞ then
3:         if L^n is t−spanned by E then
4:             return True
5:     else
6:         for i = 1, ..., q − 1 do
7:             Δx ← x_{i+1} − x_i
8:             Δy ← y_{i+1} − y_i
9:             B_w ← Δx(Δx^2 + Δy^2)^{−1/2}
10:            if B_w < 2t^{−2} − 1 then
11:                return False
12:        return True
```

Algorithm 4 constitutes a stopping criteria for Algorithm 3. Given a control set of lattice vertices E that is in increasing order of the angle inscribed by these vertices and the $x-$axis, Algorithm 4 verifies if the angle inscribed by neighboring vertices in E is sufficiently small. This is analogous to the notion of a *sufficiently narrow* wedge described in the previous section.

In the next section, we extend the notion of wedges used to define the intuitive algorithm in the previous section to discrete lattices. We also present technical results that will be used to prove the completeness of Algorithm 3.

3.3 Wedge Analysis

The proof of the correctness of Algorithm 3 relies on a system of partitioning a lattice L^n into *wedges* – sectors of $\mathbb{R}^2$ lying between straight lines passing through the origin. The high level idea is that vertices are iteratively added to the control set E in Line 5 of Algorithm 3, thus splitting each wedge in the partition of L^n into sub-wedges. We show that adding the shortest non-$t-$reachable vertex in each wedge to E results in a solution to the MTSCS problem for L^n. In this section, we introduce the notion of a wedge in $\mathbb{R}^2$, and prove several technical results required for the correctness of Algorithm 3. The proofs of each technical result in this section can be found in Appendix A

Definition 3.3.1 (t-Error of a Vertex)**.** Given lattice L^n, and a control set $E \subseteq L^n$, define

the *t-error* of any vertex $(x, y) \in L$ as:

$$t^E(x, y) = \frac{d^E(x, y)}{||(x, y)||}.$$

Observe that a vertex (x, y) is $t-$reachable using control set E for some value $t \geq 1$ if and only if $t^E(x, y) \leq t$. For any vertex $v = (x, y)$ in L^n, we may consider an equivalent polar representation of v. That is, we may write $v = (l, \theta)$ where $l = ||x, y||$, and $\theta = \arctan_2(y, x)$. This equivalence will be used in the remainder of this chapter.

Definition 3.3.2 (Wedge)**.** Given a lattice L^n, let $(x_1, y_1) = (l_1, \theta_1) \in L^n, (x_2, y_2) = (l_2, \theta_2) \in L^n$ with $\theta_1 \leq \theta_2$. A *wedge* $W[(x_1, y_1), (x_2, y_2)]$ in L^n is defined as

$$W[(x_1, y_1), (x_2, y_2)] = \{(x, y) = (l, \theta) \in L^n : \theta \in [\theta_1, \theta_2]\}.$$

Intuitively, a wedge is the set of all lattice vertices lying between two lines: the first passing through the origin and (x_1, y_1), the second passing through the origin and (x_2, y_2). An example wedge is illustrated in Figure 3.3 (Left).

Definition 3.3.3 (Deletable Wedges)**.** Given a lattice L^n and a control set E, a wedge $w \subseteq L^n$ is called *deletable* using E if every vertex in w is $t-$reachable using E.

Definition 3.3.4 (Boundary Expressible Wedge (BEW))**.** Given a lattice L^n, a wedge $w = W[(x_1, y_1), (x_2, y_2)]$ in L^n is a *boundary expressible wedge* (BEW) if, for all $(x, y) \in w$, there exists $k_1, k_2 \in \mathbb{N}$ with

$$(x, y) = k_1(x_1, y_1) + k_2(x_2, y_2).$$

The *interior* of w, denoted w^o, is the set of all vertices in w for which $k_1, k_2 \in \mathbb{N}_{>0}$. The vertices $(x_1, y_1), (x_2, y_2)$ are called the *boundary vertices* of w.

Figure 3.3 (Right) illustrates an example of the boundary expressible wedge $W[(2, 1), (1, 1)]$. We note that for any vertex (x, y) in the wedge, there exist natural numbers k_1, k_2 such that $(x, y) = k_1(2, 1) + k_2(1, 1)$.

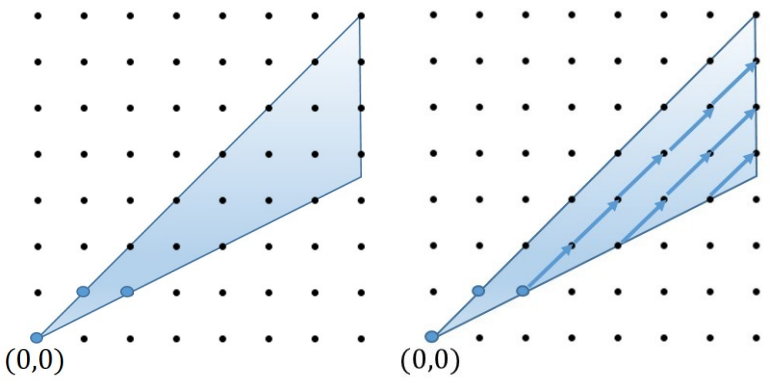


Figure 3.3: (Left) The wedge $W[(2,1),(1,1)]$ with boundary vertices $(2,1),(1,1)$ is the set of all vertices lying in the shaded blue region. (Right) The boundary expressible wedge $W[(2,1),(1,1)]$

Definition 3.3.5 (E-Bounded BEW)**.** We say that a boundary expressible wedge w is *bounded by* control set E is the boundary vertices of w are in E, but that E contains no vertices in the interior of w.

In order to prove Algorithm 3, we will show that at each iteration of the loop in line 3, the set of all lattice vertices that are not yet $t-$reachable from the iteratively constructed set E, lie in a BEW w whose boundary vertices are in E, but whose interior contains not vertices in E. We therefore prove several important Lemmas concerning these BEWs.

Lemma 3.3.6 (BEW Paths Using E)**.** *Let $t \geq 1$, and $w = W[(x_1,y_1),(x_2,y_2)]$ be a BEW that is bounded by a control set E for a lattice L^n. For any $v \in w$ if $k_1,k_2 \in \mathbb{N}$ such that $v = (k_1x_1 + k_2x_2, k_1y_1 + k_2y_2)$, then*

$$d^E(v) = k_1||x_1,y_1|| + k_2||x_2,y_2||. \tag{3.3}$$

Further, if E^* is any control set, then for all $v \in w$,

$$\left(d^{E^*}(v) < d^E(v)\right) \implies \left(\exists u \in E^* \cap w^{\mathrm{o}}\right). \tag{3.4}$$

Consider a BEW w with vertex $v \in w$ that is bounded by a control set E for a lattice L^n. Observe that Lemma 3.3.6 implies that if vertex $u \in L^n$ is such that $u, v-u \notin w$, then $d^E(v) \leq ||u|| + ||v-u||$.

The remainder of this chapter uses the following notation. Given a BEW, $w = W[(x_1,y_2),(x_2,y_2)]$ with $(x_1,y_1) = (l_1,\theta_1),(x_2,y_2) = (l_2,\theta_2)$, and a vertex $(x,y) \in w$,

we let

$$m_w(x,y) = \frac{k_2 l_2}{k_1 l_1}, \quad \theta_w = \theta_2 - \theta_1, \quad B_w = \cos(\theta_w),$$

In essence, $m_w(x,y)$ represents the slope of the vertex (x,y) relative to the boundary vertices of the wedge w. Observe that if $(x_1,y_1) = (1,0), (x_2,y_2) = (0,1)$ then $m_w(x,y)$ reduces to the familiar definition of a the slope of the vertex (x,y). With these definitions in mind, we make the following observations:

Lemma 3.3.7. *Let w be a BEW on a lattice L^n that is bounded by a control set E. Let u, v be two vertices in w. Then $m_w(u) \geq m_w(v)$ if and only if u lies above the line passing through v and the origin.*

Lemma 3.3.8 ($t-$Error for BEWs)**.** *Let $t \geq 1$, and consider lattice L^n with control set E and BEW $w = W[(x_1,y_1),(x_2,y_2)]$. If w is bounded by E, then the $t-$error of any point $(x,y) \in w^{\text{o}}$ is given by*

$$t^E(x,y) = \frac{1 + m_w(x,y)}{\sqrt{1 + m_w(x,y)^2 + 2m_w(x,y)B_w}}. \tag{3.5}$$

If, on the other hand, $(x_1,y_1) \in E$, and $(x_2,y_2) \notin E$ is $t-$reachable using E, then

$$t^E(x,y) \leq \frac{1 + tm_w(x,y)}{\sqrt{1 + m_w(x,y)^2 + 2m_w(x,y)B_w}}, \ \forall (x,y) \in w^{\text{o}}. \tag{3.6}$$

Finally, if $(x_2,y_2) \in E$ and $(x_1,y_1) \notin E$ is $t-$reachable using E, then

$$t^E(x,y) \leq \frac{t + m_w(x,y)}{\sqrt{1 + m_w(x,y)^2 + 2m_w(x,y)B_w}}, \ \forall (x,y) \in w^{\text{o}}. \tag{3.7}$$

Lemma 3.3.8 provides expressions for the t-error of BEWs. We next use this result to investigate potentially non $t-$reachable vertices using a set E that lie in a BEW that is bounded by E.

Lemma 3.3.9 (t-Reachable Vertices in a BEW using E)**.** *Let $t > 1$ and consider a lattice L^n with control set $E \subseteq L^n$ and BEW $w = W[(x_1,y_1),(x_2,y_2)]$ that is bounded by E. Then the following are true:*

1. *If $B_w \geq (2t^{-2} - 1)$, then every vertex in w is $t-$reachable using E. If $n = \infty$, then this requirement is necessary as well as sufficient.*

2. *If not every vertex in w is $t-$reachable using E, then the set of all non $t-$reachable points (x, y) in w using E is exactly equal to*

$$\begin{aligned}\bar{w} =& \{(x,y) \in w \ : \ m_w(x,y) \in (\underline{m}_w, \overline{m}_{cirt})\}, \ \textit{where} \\ \underline{m}_w =& \frac{t\left(t(1-B_w) - \sqrt{(B_w-1)(B_w t^2 + t^2 - 2)}\right)}{t^2-1} - 1, \\ \overline{m}_w =& \frac{t\left(t(1-B_w) + \sqrt{(B_w-1)(B_w t^2 + t^2 - 2)}\right)}{t^2-1} - 1.\end{aligned} \tag{3.8}$$

The well-definedness of the expressions in (A.8) is discussed in the following Remark:

Remark 3.3.10. In Lemma 3.3.9, we observe that if not every vertex in BEW w is t-reachable using control set E, then by the first result of this lemma, it must hold that $B_w > 2t^{-2} - 1$. We observe, critically, that in this case the arguments under the square-root in (A.8) are positive. Moreover, it is trivial to show that $\overline{m}_w \geq 1, \underline{m}_w \leq 1$ using the identity $B_w > 2t^{-2} - 1$.

Lemma 3.3.9 will not only help establish the correctness of Algorithm 4, but also allows us to classify wedges in a way whose convenience will become apparent. This classification is presented in the following definition.

Definition 3.3.11 (Wedge Separability)**.** Let $t > 1$, and E a control set of L^n. Consider a BEW $w = W[(x_1, y_1), (x_2, y_2)]$ that is bounded by E, and the shortest vertex in the interior of w, given by $v = (x_1 + x_2, y_1 + y_2)$ with $l_1 = ||x_1, y_1||, l_2 = ||x_2, y_2||$. Then,

- If $m_w(v) \in (\underline{m}_w, \overline{m}_w)$ – that is, if v is not $t-$reachable using E by Lemma 3.3.9 – then w is called *shortest separable*.
- If $m_w(v) \geq \overline{m}_w$, we say that w is *upper separable*. In this case, by Remark 3.3.10, we note that $m_w(v) = {}^{l_2}/_{l_1} \geq \overline{m}_w \geq 1$ implying that $l_2 \geq l_1$. If k is the smallest natural number such that ${}^{l_2}/_{kl_1} \leq \overline{m}_w$, we say that k is the *upper separability factor* of w.
- Finally, if $m_w(v) \leq \underline{m}_w$, w is said to be *lower separable*. By Remark 3.3.10, we note that $m_w(v) = {}^{l_2}/_{l_1} \leq \underline{m}_w \leq 1$ implying that $l_2 \leq l_1$. If k is the smallest natural number such that ${}^{kl_2}/_{l_1} \geq \underline{m}_w$, we say that k is the *lower separability factor* of w.

Observe, trivially, that any BEW w that is bounded by a control set E is either deletable, shortest separable, upper separable or lower separable. Thus the classification

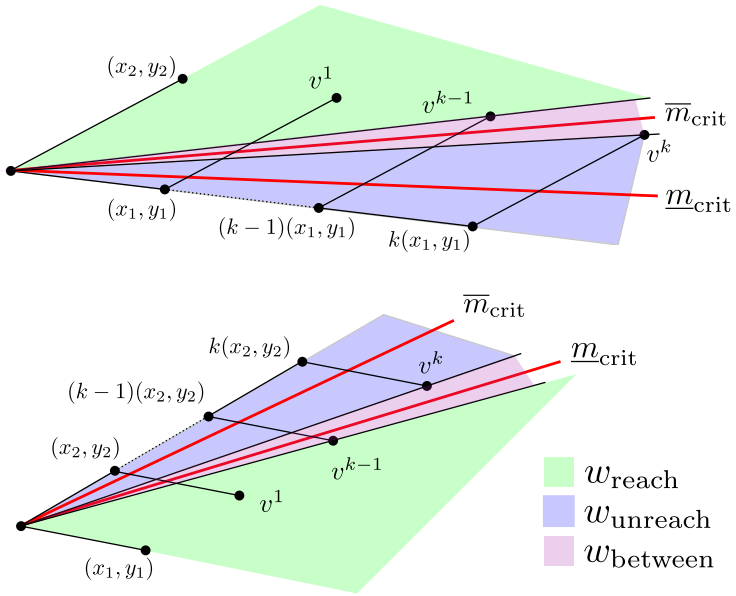


Figure 3.4: Notation for upper (top) and lower (bottom) separable wedges.

system presented in the previous definition is exhaustive for such wedges. Observe further that a wedge that is shortest separable using E can be thought of as being both upper and lower separable with upper/lower separability factor $k = 1$. The remainder of this chapter uses the following notation for wedges that are not deletable using E: for a BEW $w = W[(x_1, y_1), (x_2, y_2)]$ that is bounded by a control set E, let

$$
\begin{aligned}
v_w^1 &= (x_1 + x_2, y_1 + y_2), \\
v_w^r &= \begin{cases} (rx_1 + x_2, ry_1 + y_2) \text{ if } w \text{ is upper separable,} \\ (x_1 + rx_2, y_1 + ry_2) \text{ if } w \text{ is lower separable.} \end{cases}, \ \forall r \geq 1 \\
w_{\text{reach}} &= \begin{cases} W[v_w^{k-1}, (x_2, y_2)] \text{ if } w \text{ is upper separable with factor } k, \\ W[(x_1, y_1), v_w^{k-1}] \text{ if } w \text{ is lower separable with factor } k. \end{cases} \\
w_{\text{unreach}} &= \begin{cases} W[(x_1, y_1), v^k] \text{ if } w \text{ is upper separable with factor } k, \\ W[v_w^k, (x_2, y_2)] \text{ if } w \text{ is lower separable with factor } k. \end{cases} \\
w_{\text{between}} &= \begin{cases} W[v_w^k, v_w^{k-1}] \text{ if } w \text{ is upper separable with factor } k, \\ W[v_w^{k-1}, v_w^k] \text{ if } w \text{ is lower separable with factor } k. \end{cases}
\end{aligned}
\tag{3.9}
$$

An example using this notation can be found in Figure 3.4. We present three technical results.

Lemma 3.3.12 (BEW Closed Under Splitting)**.** *Let $w = W[(x_1, y_1), (x_2, y_2)]$ be a BEW that is bounded by a control set E for a lattice L^n. The wedges $w_1 = W[(x_1, y_1), v_w^1]$ and $w_2 = W[v_w^1, (x_2, y_2)]$ are both BEWs that are bounded by $E \cup \{v_w^1\}$, and $w_1 \cup w_2 = w$. Further, if w is upper or lower separable, then the wedges $w_{unreach}, w_{between}$ are BEWs and $w_{unreach}$ is bounded by $E \cup \{v_w^k\}$.*

As the title of the previous lemma suggests, if a BEW is split in two via the addition of v_w^1 or v_w^k, then one or both the resulting sub-wedges will be BEWs. It will be shown in the next lemma, that any sub-wedge that is not a BEW when split in this way, will be deletable. Thus, at each iteration of Algorithm 3, each vertex that is not yet $t-$reachable using the computed control set will lie in a BEW bounded by the control set.

Lemma 3.3.13. *Let $t \in (1, \sqrt{2})$, and let $w = W[(x_1, y_1), (x_2, y_2)]$ be a BEW that is bounded by a control set E for a lattice L^n. If w is upper (or lower) separable using E, then*

$$B_{w_{between}} \geq \frac{1}{t}. \tag{3.10}$$

Lemma 3.3.14. *Let $w = W[(x_1, y_1), (x_2, y_2)]$ be a BEW that is bounded by a control set E on a lattice L^n. If w is not deletable using E, let v denote the shortest non $t-$reachable vertex in w using E. The the following hold:*

1. *If w is shortest separable using E, then $v_w^1 = v$.*
2. *If w is either upper or lower separable using E and v_w^k is not $t-$reachable using E, then $v = v_w^k$ and every vertex in $w_{reach} \cup w_{between}$ is $t-$reachable using $E \cup \{v_w^k\}$.*
3. *If w is either upper or lower separable using E and v_w^k is $t-$reachable using E, then w is deletable using $E \cup \{v\}$.*

For the remainder of this chapter, let E^r denote the control set at the beginning of iteration r of the for loop in Line 3 of Algorithm 3.

Lemma 3.3.15. *If $v \in L^n$ is not $t-$reachable using E^r, then v lies in a BEW that is bounded by E^r for any $r \geq 0$.*

Lemma 3.3.16. *Let $w = W[(x_1, y_1), (x_2, y_2)]$ be a BEW that is bounded by a control set E. If w is either upper or lower separable using E, let θ denote the angle inscribed by the wedge $w_{between} \cup w_{reach}$. Then*

$$\cos(\theta) \geq \frac{2}{t^2} - 1. \tag{3.11}$$

Further, if $m_w(v_w^k) \notin (\underline{m}_w, \overline{m}_w)$*, let* v *denote the shortest vertex in* w *that is not* $t-$*reachable using* E*. Then*

$$\min(\cos(\theta_1), \cos(\theta_2)) \geq \frac{2}{t^2} - 1, \tag{3.12}$$

where θ_1, θ_2 *are the angles inscribed by the wedges* $w_1 = W[(x_1, y_1), v], W[v, (x_2, y_2)]$*, respectively.*

The above lemma implies that if v_w^k is added to a control set bounding w, then the green and pink sub-wedges in Figure 3.4 become deletable using the resulting control set. Further, this lemma implies that if v_w^k is *not* the shortest non $t-$reachable vertex in w using a control set that bounds w, then adding the shortest non $t-$reachable vertex to the control set results in w being deletable using that control set.

Lemma 3.3.17 (Distance Invariance)**.** *Let* $t \in (1, \sqrt{2})$*, and let* E^r *denote the control set at the beginning of iteration* r *of the for loop in Line 3 of Algorithm 3. Let* $w = W[(x_1, y_1), (x_2, y_2)]$ *be a BEW wedge that is bounded by* E^r*. Then the following are true:*

1. *If* w *is shortest separable using* E^r*, then for any other control set* E' *with* $v_w^1 \notin E'$*, it holds that* $d^{E'}(v_w^1) \geq d^{E^r}(v_w^1)$*.*
2. *If* w *is upper or lower separable using* E^r *and* $m_w(v_w^k) \in (\underline{m}_w, \overline{m}_w)$*, then for all vertices* $u \in w_{unreach}, u \neq v$*,* $||u|| + ||v_w^k - u|| \geq d^{E^r}(v_w^k)$*.*

Lemma 3.3.18 (Base Case For Inductive Proof)**.** *Let* $t > 1$ *and* L^n *a lattice. Let* $\mathcal{E}^t$ *denote the set of all minimal* $t-$*spanning control sets of* L^n*, and let* $E = \{(0,1), (1,0), (-1,0), (0,-1)\}$*. If* $t \geq \sqrt{2}$ *then* $E \in \mathcal{E}^t$*. Furthermore, if* $t < \sqrt{2}$ *then* $E \subseteq E^*$ *for all* $E^* \in \mathcal{E}^t$*.*

3.4 Completeness, Solution Size, & Path Error

In this section, we use the technical results of the previous section to prove the completeness of Algorithm 3. We also present bounds on the size of the control set returned by the algorithm and bound the maximum error incurred from computed motions with respect to free-space optimal.

Theorem 3.4.1 (Completeness)**.** *The Algorithm 3 is complete. That is, it returns a solution to Problem 2.3.8 for lattice* L^n *in finite time for all* $t > 1$ *and for all lattices* $L^n, n \in \mathbb{N} \cup \{\infty\}$*.*

Proof. At each iteration r of the for loop in Line 3 of Algorithm 3, let $E^r = \{(x_1, y_1), (x_2, y_2), \ldots, (x_q, y_q)\}$ denote the computed control set at the beginning of this iteration. Assume without loss of generality that the elements in E^r, $(x_i, y_i) = (l_i, \theta_i)$ are written in order of $\theta_i, i = 1, \ldots, q$. Let v^r denote the vertex in L^n that is added to E^r in Line 5 and let

$$\begin{aligned}
\mathcal{N}^r &= \{W[(x_i, y_i), (x_{i+1}, y_{i+1})], i = 1, \ldots, q-1\} \cup \{W[(x_q, y_q), (x_1, y_1)]\}, \\
\mathcal{N}^r_{\text{BEW}} &= \{w \in \mathcal{N}^r \ : \ w \text{ is a BEW}\} \\
\overline{\mathcal{N}}^r &= \{w \in \mathcal{N}^r_{\text{BEW}} \ : \ w \text{ is not deletable}\}
\end{aligned}$$

Observe that $\bigcup_{w \in \mathcal{N}^r} w = L^n$ and that wedges w in $\overline{\mathcal{N}}^r$ are bounded by E^r. By Lemma 3.3.15, v^r must lie in a wedge $w^r \in \overline{\mathcal{N}}^r$. Let $w^{r,\text{o}}$ denote the interior of w^r.

We begin by showing that Algorithm 3 returns a $t-$spanning control set if it terminates. Observe that it is sufficient to show that Algorithm 4 returns a value True given a control set E^r if and only if E^r is a $t-$spanning control set of L^n. If $n < \infty$, this result holds trivially by Lines 2-4 of Algorithm 4. If $n = \infty$, then Algorithm 4 returns true given a control set E^r if and only if $\cos(\theta) \geq 2t^{-2} - 1$ for each $w \in \mathcal{N}^r$ (Lines 7-12) where θ denotes the angle inscribed by w. By Lemma 3.3.16, each $w \in \mathcal{N}^r - \mathcal{N}^r_{\text{BEW}}$ meets this criteria. Therefore Algorithm 4 returns True given E^r if and only if each $w \in \mathcal{N}^r_{\text{BEW}}$ meets this criteria. Since each wedge is $\mathcal{N}^r - \mathcal{N}^r_{\text{BEW}}$ is deletable using E^r (Lemma 3.3.15), and each wedge $w \in \mathcal{N}^r_{\text{BEW}}$ is deletable using E^r if and only if $\cos(\theta) = B_w \geq 2t^{-2} - 1$ (Lemma 3.3.9) we conclude that Algorithm 4 returns true given E^r if and only if E^r is a $t-$spanning control set of L^n.

We now show that Algorithm 3 terminates in finite time. Observe that this holds if $n < \infty$ since, in the worst case, $E = L^n$ will $t-$span L^n for all $t \geq 1$, and L^n is of finite size. Therefore, since it was show that Algorithm 4 returns True given a control set E if and only if E is a $t-$spanning control set of L^n, it must hold that a value of True will be returned given $E = L^n$ in finite time. Assume $n = \infty$. Given an arbitrary iteration r, and an arbitrary wedge $w = W[(x_1, y_1), (x_2, y_2)] \in \mathcal{N}^r$, let $r+r_1, r+r_2, \ldots,$ denote a sequence of iterations such that $v^{r+r_i} \in w$ for all $i \geq 1$. Then it suffices to show that for all such wedges in $\mathcal{N}^r$, there exists a natural number $N < \infty$ such that w is deletable using E^{r+r_N}. At iteration $r+r_1$, w is split into sub-wedges $w_1 = W[(x_1, y_1), v^{r+r_1}], w_2 = W[(v^{r+r_1}), (x_2, y_2)]$. If $v^k_w \neq v^{r+r_1}$ (where k is the upper/lower separability factor if w is upper/lower separable and 1 if it is shortest separable), then w is deletable using E^{r+r_1} (Lemma 3.3.14) which completes the proof. Otherwise, $v^{r+r_1} = v^k_w$. By the law of cosines, if $B_{w'} = \max(B_{w_1}, B_{w_2})$,

then

$$B_{w'} = \begin{cases} \frac{m_w(v_w^k)B_w+1}{\sqrt{2m_w(v_w^k)B_w+m_w(v_w^k)^2+1}}, & \text{if } l_2 \geq l_1 \\ \frac{m_w(v_w^k)+B_w}{\sqrt{2m_w(v_w^k)B_w+m_w(v_w^k)^2+1}}, & \text{otherwise,} \end{cases} \tag{3.13}$$

where $l_1 = ||x_1, y_1||, l_2 = ||x_2, y_2||$. The first case accounts for upper separable wedges w using E^{r+r_1} and shortest separable wedges where $l_2 \geq l_1$, while the second case accounts for lower separable and shortest separable wedges where $l_2 < l_1$. Minimizing $B_{w'}/B_w$ with respect to $m_w(v_w^k)$ and observing that $m_w(v_w^k) \leq \overline{m}_w$ if $l_2 \geq l_1$ and $m_w(v_w^k) \geq \underline{m}_w$ if $l_2 < l_1$, it is easily verified that

$$\frac{B_{w'}}{B_w} \geq t, \ \forall t \in (1, \sqrt{2}]$$

Since it is assumed that $t > 1$, we can conclude that when Algorithm 3 adds a vertex to a wedge w, the value of B_w of each sub-wedge of w is increased by a factor $1 + \epsilon$ for some $\epsilon > 0$. Therefore, on iteration $r + r_i$, each sub-wedge $w' \subseteq w, w' \in \mathcal{N}^{r+r_i}$ will be such that $B_{w'} \geq (1 + \epsilon)^{r_i} B_w$. Setting $r_N = \lceil \ln((2t^{-2} - 1)B_w^{-1}) \ln(1 + \epsilon)^{-1} \rceil$ yields $B_{w'} \geq 2t^{-2} - 1$ implying that w' is deletable using E^{r+r_N} where $r_N < \infty$ which completes the proof.

The arguments above prove that Algorithm 3 will return a $t-$spanning control set in finite time. It will now be shown that this set E is of minimal size. Let $\mathcal{E}^*$ denote the set of all solutions to Problem 2.3.8, and let $E^* \in \mathcal{E}^*$. To show $|E| \leq |E^*|$ it suffices to show that there exists an injection $f : E \to E^*$. We define this function recursively on r. For each $i \in E^0$ where E^0 is given in Line 2, let $f(i) = i$. Observe that this is a valid mapping from E to E^* since $E^0 \subseteq E^*$ for all $E^* \in \mathcal{E}^*$ by Lemma 3.3.18. Next, for all $r > 0$ if vertex v^r is added to E^r in Line 5 with $v^r \in w^r \in \overline{\mathcal{N}}^r$, then

1. If w^r is shortest separable, $v^r = v_{w^r}^1$ (Lemma 3.3.14). Let $f(v^r) = v^r$. Note $v^r \in E^*$. If not, then $d^{E^*}(v_{w^r}^1) \geq d^{E^r}(v_{w^r}^1) \geq t||v_{w^r}^1||$ (Lemma 3.3.17) and E^* is not a $t-$spanning control set. Observe that $v_{w^r}^1 \in w^{r,\mathrm{o}}$.

2. If w^r is upper/lower separable and $v_{w^r}^k \in (\underline{m}_{w^r}, \overline{m}_{w^r})$, then $v^r = v_{w^r}^k$ (Lemma 3.3.14). There must exist a vertex $u \in E^* \cap \left(w_{\text{reach}}^r \cup w_{\text{between}}^r\right)$. Indeed, there must exist $u \in E^* \cap w^{r,\mathrm{o}}$ such that the path using E^* to $v_{w^r}^k$ passes through u (Lemma 3.3.6). If $u \in w^r - \left(w_{\text{reach}}^r \cup w_{\text{between}}^r\right) = w_{\text{unreach}}^r$, then $d^{E^*}(v_{w^r}^k) \geq ||u|| + ||v_{w^r}^k - u|| \geq d^{E^r}(v_{w^r}^k) \geq t||v_k^{w^r}||$ (Lemma 3.3.17) and E^* is not a $t-$spanning control set. Let $f(v^r) = u$ and recall that $u \in w^{r,\mathrm{o}}$.

3. If w is upper/lower separable and $v_{w^r}^k \notin (\underline{m}_w, \overline{m}_w)$, then by Lemma 3.3.6, there must exist a vertex $u \in w^{r,\mathrm{o}} \cap E^*$ since v is not t-reachable using E^r. Let $f(v^r) = u$.

To show that f is injective, observe that f maps $v^r \in w^r$ to a vertex $u \in w^{r,\mathrm{o}}$ where $w^r \in \overline{\mathcal{N}}^r$. Therefore, to show injectivity, it suffices to show that $u \notin w^{s,\mathrm{o}}$ for any wedge $w^s \in \overline{\mathcal{N}}^s$ on any iteration $s > r$.

If w^r is shortest separable, then $f(v^r) = v^r = v^1_{w^r}$. Since $v^1_{w^r}$ lies on the boundary of the wedges $W[(x_1, y_1), v^1_{w^r}], W[v^1_{w^r}, (x_2, y_2)] \in \mathcal{N}^{r+1}_{\mathrm{BEW}}$ (Lemma 3.3.12), v^r will never appear in the interior of any $w^s \in \mathcal{N}^s_{\mathrm{BEW}}$ on any other iteration $s > r$. Therefore, v^r will never appear in the interior of any $w^s \in \overline{\mathcal{N}}^s \subseteq \mathcal{N}^w_{\mathrm{BEW}}$.

If w^r is upper/lower separable with $v^k_{w^r} \in (\underline{m}_{w^r}, \overline{m}_{w^r})$, then $f(v^r) = u$ for $u \in w^r_{\mathrm{reach}} \cup w^r_{\mathrm{between}}$. Since $w^r_{\mathrm{reach}} \cup w^r_{\mathrm{between}}$ is deletable using $E^r \cup \{v^r\}$ (Lemma 3.3.14), $(w^r_{\mathrm{reach}} \cup w^r_{\mathrm{between}}) \cap w^{s,\mathrm{o}} = \emptyset$ for any $w^s \in \overline{\mathcal{N}}^s$ on any iteration $s > r$ implying that $u \notin w^{s,\mathrm{o}}$.

Finally, if w^r upper/lower separable using E^r with $v^k_{w^r} \notin (\overline{m}_{w^r}, \underline{m}_{w^r})$, then w^r is deletable using $E^r \cup \{v^r\}$ (Lemma 3.3.14) implying that $w^r \cap w^{s,\mathrm{o}} = \emptyset$ for any $w^s \in \overline{\mathcal{N}}^s$ on any iteration $s > r$ implying that $u \notin w^{s,\mathrm{o}}$

Therefore, f is an injective function from E to E^* implying that $|E| \leq |E^*|$ implying in turn that E is a $t-$spanning control set of minimal size. □

We now present a bound on the size of the control set returned by Algorithm 3.

Theorem 3.4.2 (Control Set Size)**.** *Let E denote the size of the control set returned by Algorithm 3 given a lattice L^n and value $t > 1$. Then,*

$$|E| = \begin{cases} \Theta\left(\sqrt{\frac{t}{t-1}}\right), & \text{if } n = \infty, \\ \min\left(O\left(\sqrt{\frac{t}{t-1}}\right), O(n^2)\right), & \text{otherwise.} \end{cases} \tag{3.14}$$

Proof. We begin by showing that $|E| = \Omega((t-1)^{-1/2})$ if $n = \infty$. Assume that $E = \{(x_1, y_1), (x_2, y_2), \ldots, (x_q, y_q)\}$ where, without loss of generality, the elements in E, $(x_i, y_i) = (l_i, \theta_i)$ are written in order of $\theta_i, i = 1, \ldots, q$. By the first result in Lemma 3.3.9 and Lemma 3.3.16, if E is a $t-$spanning control set, then for each $(x_i, y_i), (x_{i+1}, y_{i+1}) \in E$, $\cos(\theta_{i+1} - \theta_i) \geq 2t^{-2} - 1$. Therefore,

$$|E| \geq \frac{\pi}{\arccos\left(\frac{2}{t^2} - 1\right)} = \Theta\left(\sqrt{\frac{t}{t-1}}\right),$$

implying that $|E| = \Omega((t-1)^{-1/2})$. Next we show that $|E| = O((t-1)^{-1/2})$ for all $n \leq \infty$. Consider the set $E' = \{(x_1, y_1), (x_3, y_3), \ldots, (x_{2\lfloor q/2 \rfloor + 1}, y_{2\lfloor q/2 \rfloor + 1})\}$. Then, $\cos(\theta_{2j+1} - \theta_{2j-1}) < 2t^{-2} - 1$ for all $j = 1, \ldots \lfloor q/2 \rfloor$. Indeed, if this were not the case, then the wedge

$W[(x_{2j-1}, y_{2j-1}), (x_{2j+1}, y_{2j+1})]$ would be deletable using $E - \{(x_{2j}, y_{2j})\}$ By the first result of Lemma 3.3.9, and Lemma 3.3.16 implying that E is not of minimal size. This is a contradiction of the result of Theorem 3.4.1. Therefore,

$$|E'| \leq \frac{\pi}{\arccos(2t^{-2} - 1)} = \Theta\left(\sqrt{\frac{t}{t-1}}\right).$$

Observe that $|E| = O(|E'|)$. Therefore, $|E| = O\left(\sqrt{\frac{t}{t-1}}\right)$ and $|E| = \Theta\left(\sqrt{\frac{t}{t-1}}\right)$ if $n = \infty$. Finally, suppose $n < \infty$. As t approaches 1, the control set size will approach $(24/\pi^2)n^2 + O(n\log(n)) = O(n^2)$ [105]. □

Next, we will present a bound on the error incurred by using a MTSCS to plan a path. We begin a definition.

Definition 3.4.3 (δ-Robustly Feasible Path)**.** Let $\mathbb{B}_v(r) \subset \mathbb{R}^2$ denote an open ball of radius r centered at $v \in \mathbb{R}^2$. Given a set of obstacles $\mathcal{X}_{\text{obs}} = \{O_1, \ldots, O_m\}$ and a value $\delta > 0$, a path $\pi : [0, s_f] \to \mathbb{R}^2$ with final arc-length s_f is said to be *δ-robustly feasible* if $\mathbb{B}_{\pi(s)}(\delta) \cap O_i = \emptyset$ for all $s \in [0, s_f]$ and for all $O_i \in \mathcal{X}_{\text{obs}}$.

An illustrative example of a $\delta-$robustly feasible path is illustrated in Figure 3.5

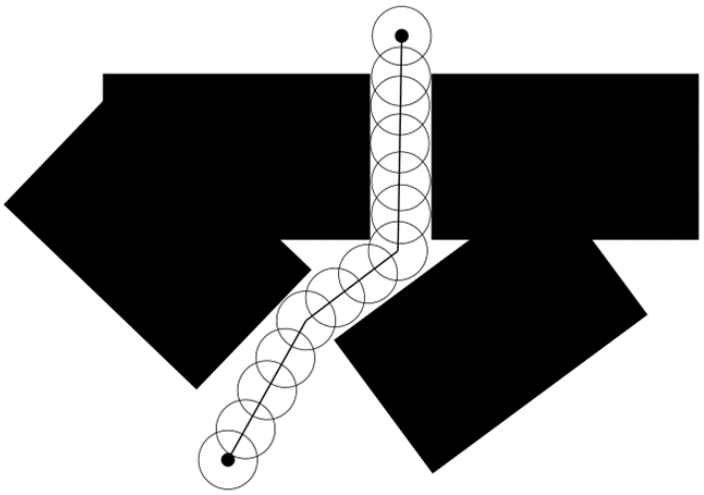


Figure 3.5: Example of a δ-robustly feasible path.

The lattice L^n in (3.1) is defined for integer vertices implying that the distance between a lattice vertex and its closest neighbor in the lattice is 1. However, this distance *1 lattice unit* is given without actual units of measurement. For actual units of measurement, say meters, 1 lattice unit may represent 1 meter, half of a meter, etc. We define the *fineness* of a lattice Δ as the measurement of 1 lattice unit given units of measurement. Observe that if a lattice L^n has fineness Δ, then for *any* configuration $v \in \mathcal{X}$, it must hold that there is a lattice vertex a distance of no more than $\sqrt{2}\Delta$ from v. Using these definitions, we present an error bound on path length.

Theorem 3.4.4 (Path Error)**.** *Let $\mathcal{X}_{obs}$ be a set of obstacles, let $\delta > 0$, and let π^* denote the shortest δ-robustly feasible path from a start vertex p_s to a goal vertex p_g in the lattice L^n. For a lattice L^n defined in (3.1) with fineness Δ, let E be a MTSCS for L^n with a value $t \in (1, \sqrt{2}]$. Finally, let π^E be the shortest path in the graph $G_{CF} = (L^n, \bar{E}_{CF}, ||\cdot||)$ from p_s to p_g where $\bar{E}_{CF}$ is defined in (2.7) for the control set E. If*

$$\delta \geq 2\sqrt{2}\Delta$$

then

$$\frac{c(\pi^E)}{c(\pi^*)} \leq t\left(1 + \frac{2\sqrt{2}\Delta}{\delta - 2\sqrt{2}\Delta}\right) \tag{3.15}$$

In [50, Theorem 2], a similar bound is presented with condition $\delta > 3\sqrt{2}\Delta$ and error ratio $r = (1 + {}^{2\sqrt{2}\Delta}/_{(\delta - 3\sqrt{2}\Delta)})$. Observe that the error ratio in 3.15 outperforms r for small values of t and larger values of Δ, but fails to outperform r for larger values of t.

Proof. The proof of this Theorem is a modified version of the proof of [50, Theorem 2], altered to accommodate $t-$factor sub-optimality. We begin with a sub-claim: Let π be a straight line segment connecting lattice vertices v_1, v_2 with length l, and let π_t^E be the path between v_1, v_2 obtained using a MTSCS E in the absence of obstacles. Let l^E be the length of π_t^E, and observe $l^E \leq tl$. Finally, let D denote the maximum lateral error between π and π_t^E. Then

$$D \leq \frac{l}{2}\sqrt{t^2 - 1}. \tag{3.16}$$

Indeed, let v denote the configuration in π_t^E at which D is realized. Maximizing D subject to the constraints $d^E(v_1, v) + d^E(v, v_2) = l^E \leq tl$, we obtain that D is maximized by a straight line segments connecting v_1 to v and v to v_2 and $d^E(v_1, v) = d^E(v, v_2) = tl/2$. The result 3.16 follows from Pythagoras.

Next we make another sub-claim: if π is a line segment connecting v_1, v_2 with length $l < 2\delta$, if $\mathbb{B}_\delta(v_1) \cap O = \emptyset, \mathbb{B}_\delta(v_2) \cap O = \emptyset, \forall O \in \mathcal{X}_{\text{obs}}$, and if D denotes the maximum lateral error between π and a configuration v, then

$$\left(D \leq \sqrt{\delta^2 - (l/2)^2}\right) \implies (v \cap O = \emptyset, \forall O \in \mathcal{X}_{\text{obs}}). \tag{3.17}$$

This result follows immediately from Pythagoras noting that the intersection point of two circles with equal radii δ whose centers are l apart will occur a distance of $\sqrt{\delta^2 - (l/2)^2}$ from the line connecting the centers of the circles.

Observe that if $c(\pi^*) \leq \delta - 2\sqrt{2}\Delta$, then let L be the straight line segment connecting p_s, p_g with length l and observe that L lies within a ball of radius δ of p_s since $l \leq c(\pi^*) < \delta$. Therefore, the path π^E will have a maximum value of D of

$$D \leq \frac{c(\pi^*)}{2} < \delta,$$

for all $t \in (1, \sqrt{2}]$, implying that π^E does not collide with an obstacle. In this case, $c(\pi^E)/c(\pi^*) \leq t$ and 3.16 holds. Otherwise, if $c(\pi^*) > \delta - 2\sqrt{2}\Delta$, we partition π^* into N segments π_i^* with endpoints x_i, x_{i+1} for $i = 0, \ldots, N-1$. For simplicity, let $x_0 = x_s, x_N = x_g$. We form this partition in such a way as to ensure that the first $N-1$ segments have length $\delta - 2\sqrt{2}\Delta$ and the final segment has length $\leq 2(\delta - 2\sqrt{2}\Delta)$. That is, there are $\lfloor c(\pi^*)/(\delta - 2\sqrt{2}\Delta) \rfloor - 1$ segments of length $\delta - 2\sqrt{2}\Delta$ and 1 segment of length $\leq 2(\delta - 2\sqrt{2}\Delta)$ for a total of $N = \lfloor c(\pi^*)/(\delta - 2\sqrt{2}\Delta) \rfloor$ segments.

By the definition of the lattice fineness Δ, for each segment π_i^*, $i = 0, \ldots, N-2$ between x_i, x_{i+1}, it must hold that there are lattice vertices x_i', x_{i+1}' with $||x_i - x_i'||, ||x_{i+1} - x_{i+1}'|| \leq \sqrt{2}\Delta$. Therefore, letting π_i' be the line segment connecting x_i', x_{i+1}' with length l_i', and π_i^E the path using E between x_i', x_{i+1}' with length l_i^E, we can conclude that $l_i' \leq \delta$ and $d^E(x_i', x_{i+1}') \leq t l_i'$. The first holds because the length of the path π_i^* is $\delta - 2\sqrt{2}\Delta$ and l_i' is at most $l_i + 2\sqrt{2}\Delta$. While the second holds by the definition of a $t-$spanning control set. Observe that the maximum lateral error between π_i' and π_i^* is $\sqrt{2}\Delta$. Therefore, the maximum lateral error D^E between π_i^E and π_i^* is bounded by $\sqrt{2}\Delta + D$ where D is the maximum lateral error between π_i^E and π_i'. From (3.16),

$$D^E \leq \sqrt{2}\Delta + \frac{l_i'}{2}\sqrt{t^2 - 1} \leq \sqrt{2}\Delta + \frac{\delta}{2}\sqrt{t^2 - 1}.$$

The configurations x_i, x_{i+1} lie on the path π_i^* which has strong δ-clearance. Therefore, $\mathbb{B}_\delta(x_i) \cap O = \mathbb{B}_\delta(x_{i+1}) \cap O = \emptyset, \ \forall O \in \mathcal{X}_{\text{obs}}$. Further, the maximum lateral error between

all configurations lying on the path π_i^E is given by D^E above. Since the length of π_i^* is $\delta - 2\sqrt{2}\Delta$ which is at least the length of the line segment connecting x_i, x_{i+1}, we can conclude by (3.17), that

$$\sqrt{2}\Delta + \frac{\delta}{2}\sqrt{t^2-1} \leq \sqrt{\delta^2 - (\delta - 2\sqrt{2}\Delta)^2/4} \implies \pi_i^E \cap O = \emptyset, \forall O \in \mathcal{X}_{\text{obs}}.$$

The left hand side of the implication holds for all $t \in (1, \sqrt{2}]$ provided that $\delta \geq 2\sqrt{2}\Delta$. This analysis shows that π_i^E does not collide with an obstacle for any $i = 0, \ldots, N-2$. A similar analysis can be used to show that the final segment is also collision free, observing that the length of the final segment is at most $2(\delta - 2\sqrt{2}\Delta)$ but the maximum difference between the length of π_{N-1}^* and l'_{N-1} is $\sqrt{2}\Delta$ instead of $2\sqrt{2}\Delta$ since both π_{N-1}^* and π'_{N-1} terminate at the same lattice vertex. We may therefore conclude that the path π^E formed by connecting segments π_i^E does not collide with an obstacle. Therefore the shortest such path has cost no more than the path constructed above. Thus,

$$\begin{aligned} c(\pi^E) &= \sum_{i=0}^{N-1} c(\pi_i^E) \leq t \sum_{i=0}^{N-1} c(\pi'_i) \\ &= t\left(c(\pi'_0) + c(\pi'_{N-1}) + \sum_{i=1}^{N-2} c(\pi'_i)\right) \\ &\leq t\left(c(\pi_0^*) + c(\pi_{N-1}^*) + (N-1)2b + \sum_{i=1}^{N-2} c(\pi_i^*)\right) \\ &= t(c(\pi^*) + 2(N-1)b). \end{aligned} \tag{3.18}$$

Recalling that $N = \lfloor c(\pi^*)/(\delta - 2\sqrt{2}\Delta) \rfloor$ completes the proof. □

3.5 Evaluation

In this section, we compare our work against the state of the art in any-angle path planning. The Algorithms described here were encoded in Python 3.7 (Spyder). Results were obtained using a desktop equipped with an AMD Ryzen 3 2200G processor and 8GB of RAM running Windows 10 OS.

3.5.1 Computing a Control Set

We start by comparing our technique against a current state of the art control set generation method [94] which we denote SOA (state of the art). In this work, the authors propose a method to compute a control set of size 2^k for an input value $k \in \mathbb{N}_{\geq 2}$ that is based on the Fibonacci numbers. In detail, the technique begins like Algorithm 3 with a set E as in Line 2. It then adds vertex v^1 (given in (3.9)) to each wedge repeatedly until a control set of size 2^{k+2} is reached. This technique over-emphasizes wedges that already feature low $t-$errors and provides no stopping criteria other than set size.

Let E^k_{SOA} denote the set returned by the SOA algorithm for a given input k, and let E^t_P denote the set returned by Algorithm 3 for an infinite lattice and value $t > 1$. We compare E^k_{SOA}, E^t_P via two metrics. First, given a value $k \in \mathbb{N}_{\geq 2}$, we compute the smallest value of t for which E^k_{SOA} is a $t-$spanning control set of the lattice. We use this value of t as input to Algorithm 3 to compute E^t_P and compare $|E^k_{\text{SOA}}|/|E^t_P|$. Though we perform this comparison numerically, an analytic bound on the ratio can be obtained. Indeed, it can be shown that for any $k \in \mathbb{N}_{\geq 0}$, the longest vertex in E^k_{SOA} will be $(k, 1)$ and the smallest value of t such that E^k_{SOA} is a $t-$spanning control set is

$$\bar{t} = \sqrt{\frac{2\sqrt{k^2+1}}{\sqrt{k^2+1}+k-2}}.$$

Therefore, by Theorem 3.4.2,

$$\frac{|E^k_{\text{SOA}}|}{|E^{\bar{t}}_P|} = \Omega\left(\frac{2^{k+2}\sqrt{\bar{t}-1}}{\sqrt{\bar{t}}}\right) = \Omega\left(\frac{2^k}{3k}\right).$$

This result implies that E^k_{SOA} will be exponentially larger than E^t_P for the same error. Figure 3.6 (top) summarizes our findings. A visual comparison of the two control sets E^k_{SOA}, E^t_P can be found in Figure 3.7 for values of k ranging from 1 to 10. While the control sets are equal for $k = 1, 2$, the a notable difference arises for larger values of k. Observe that E^t_P is more akin to the intuitive control set for continuous space given in Figure 3.1 (d) than E^k_{SOA}

Next we compare $t\text{Er}(E^k_{\text{SOA}}) = t_{\text{SOA}}$ and $t\text{Er}(E^t_P) = t_P$ given a fixed control set size. Here the $t-$error of a control set is defined in Definition 2.3.6. In detail, for a fixed value $k \in \mathbb{N}_{\geq 2}$, compute the lowest value of t such that E^k_{SOA} $t-$spans the lattice. Next we compute the smallest value of t such that E^t_P $t-$spans the lattice subject to the constraint $|E^t_P| \leq 2^k$. The results can be seen in Figure 3.6 (bottom).

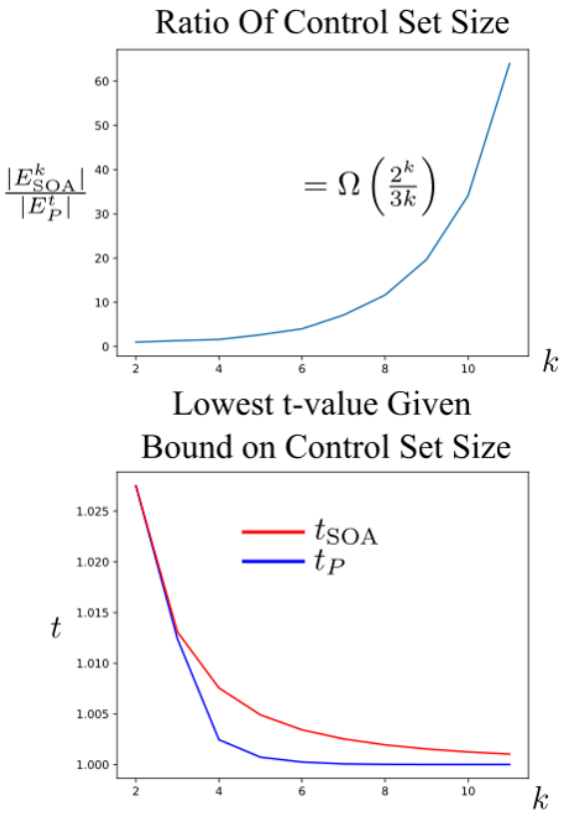


Figure 3.6: Comparison of two metrics for E^t_P and E^k_{SOA}.

3.5.2 Any-Angle Path Planning

In this section, we compare lengths of paths computed using the proposed control set against commonly used methods for any-angle path planning.

We begin with a value $k \in \mathbb{N}_{\geq 2}$ from which we compute the set E^k_{SOA} described in the previous section. The minimum value of t for which E_{SOA} is a MTSCS of the lattice is computed, and used as input for Algorithm 3 for an infinite lattice. E^k_{SOA} and E^t_P were

then used to define graphs in which a shortest path was computed between start and goal vertices. In this section, we use the basic A* algorithm in Algorithm 1 with the heuristic of a vertex given by the euclidean distance between the vertex and the goal in the absence of obstacles. We label the

We compare our methods against five common algorithms: three sampling algorithms

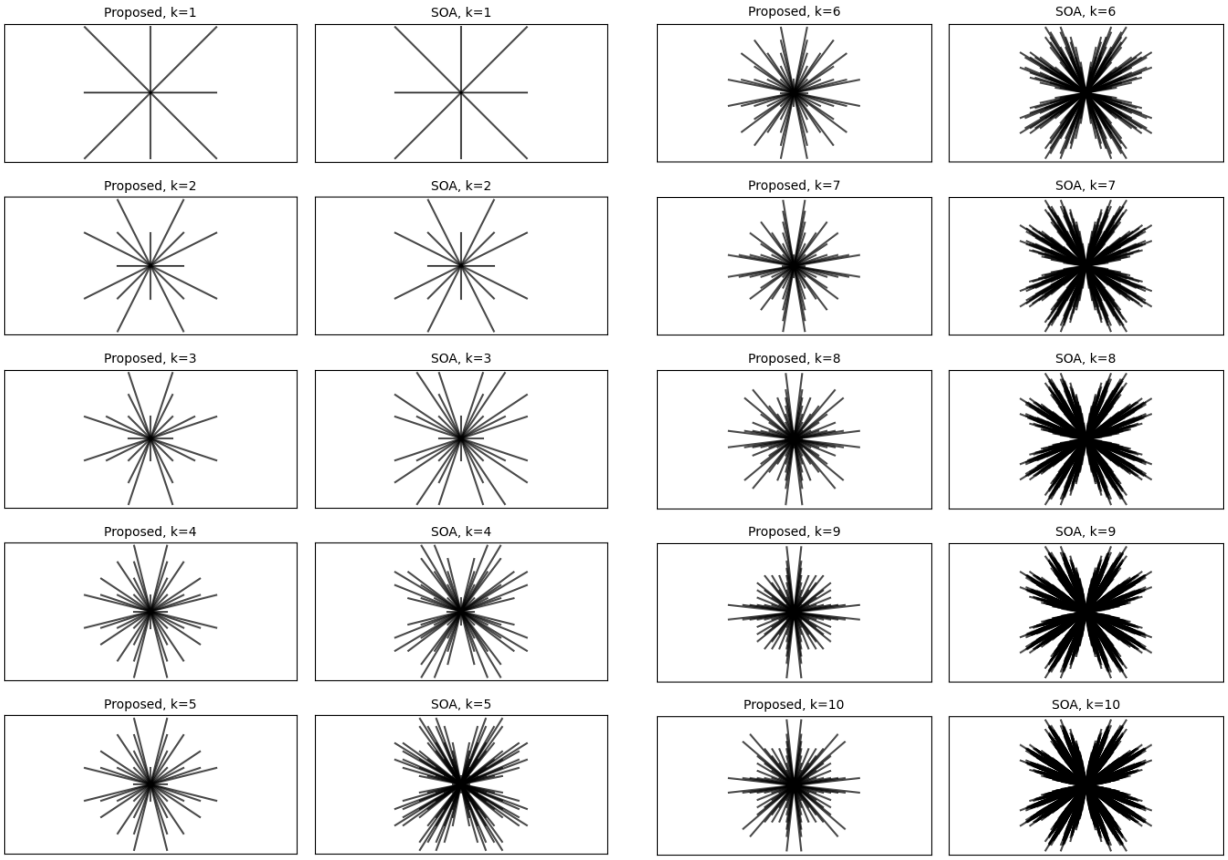
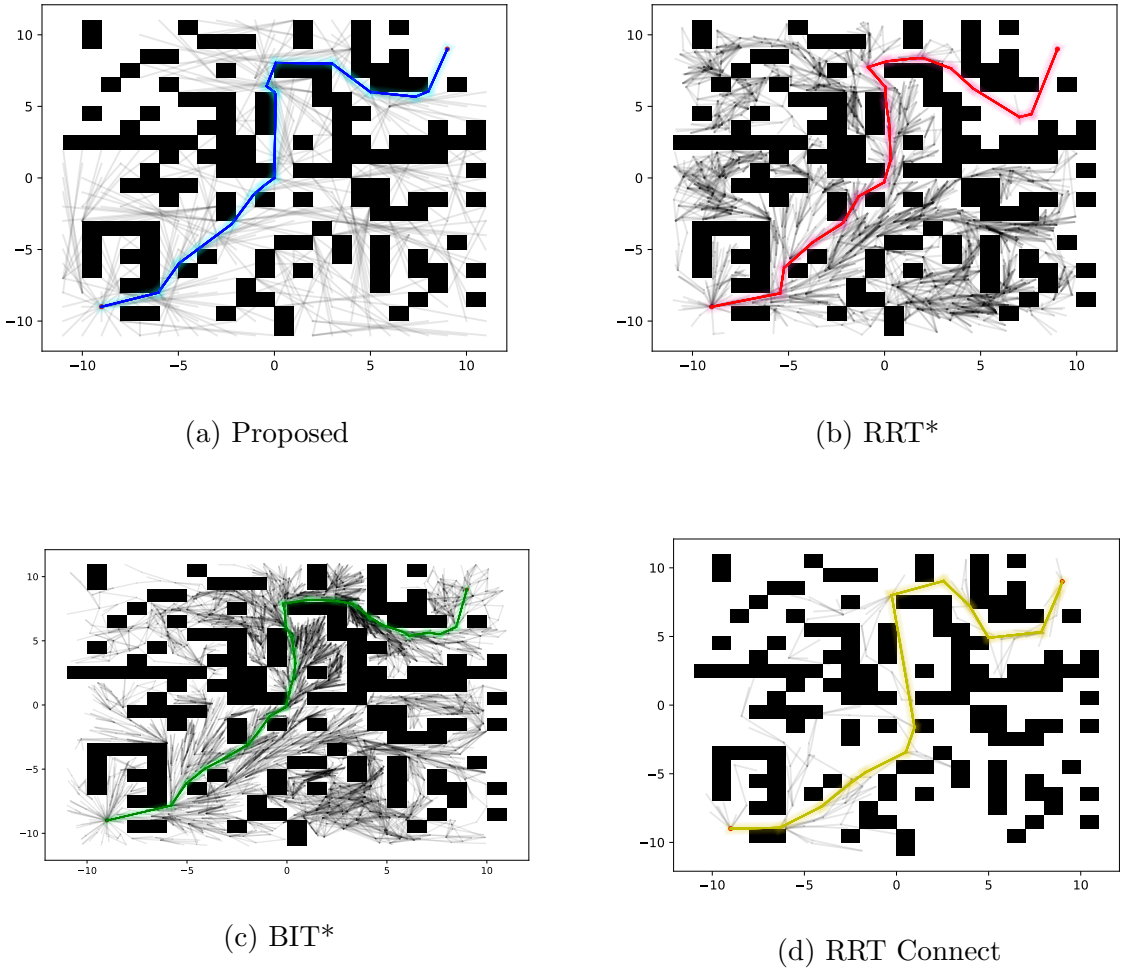


Figure 3.7: Visual comparison of E_p^t, E_{SOA}^k for varying k.

– asymptotically optimal Rapidly exploring Random Trees (RRT*) [54], Batch Informed Trees (BIT*) [32], and RRT Connect [63] – and two line-of-sight based algorithms – ANYA [43] and its improved version labelled here as iANYA [44]. Figure 3.8 compares the proposed method against the three sampling planners for one such planning problem. Here near-optimality was reached by the proposed method with a value of $t \approx 1.001$. The fastest algorithm, on average by far, is RRT Connect, exploring very few configurations before a connection is made. However, once this connection is made, the path is returned. The remaining two algorithms are asymptotically optimal and constitute *any-time* planners. That is, an initial path is quickly computed and then refined up to a specified time cut-off. For a fixed time, we compare the proposed method against RRT*, BIT*, ANYA, and iANYA.

In detail, 500 instances of the motion planning problem were generated by randomly

selecting an obstacle set $\mathcal{X}_{\text{obs}}$ and start-goal lattice vertices. These we problems were then solved using A* in conjunction with control sets E^k_{SOA}, E^t_P. The run time of the former was measured and used as a time cut-off for the any-time sampling planners.



(a) Proposed

(b) RRT*

(c) BIT*

(d) RRT Connect

Figure 3.8: Visualization of planning methods

Figure 3.9 summarizes our findings. Algorithm iANYA – shown in black – serves as a baseline since it is known to be optimal. As k increases (and therefore as the run-time cut-off increases), BIT* converges to optimal faster than RRT*. This is not surprising since BIT* makes use of a heuristic to guide random sampling. Notably, the proposed algorithm

out-performs both RRT* and BIT* given a fixed run time cut-off. Further, the proposed method reaches near optimal for a value $k = 4$. In terms of run time, the proposed method outperformed ANYA up to a value of $k = 5$ on average. However, the proposed method was outperformed both in path length and runtime by iANYA for all values of $k \geq 1$. Even with $k = 1$, iANYA required on average 12% less time than the proposed method.

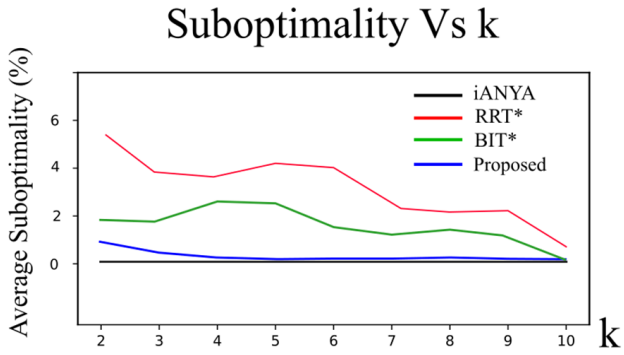


Figure 3.9: Sub-optimality of compared methods.

3.6 Discussion

In this work, we showed that the MTSCS problem can be efficiently solved for a certain class of problem with lattices of the form (3.1). Notably, for any value $t > 1$, there exists a finite MTSCS even for an infinite lattice. Bounds on the size of the MTSCS as well as on the error incurred were presented. The resulting control sets have notable benefits over the current state of the art including exponentially smaller sizes for the same $t-$factor sub-optimality, and smaller $t-$factor sub-optimality given fixed control set size. However, for any-angle path planning in a 2-dimensional workspace, our methods fail to outperform iANYA, an algorithm enjoying both optimalality and run time efficiency. However, as graph-search methods improve, the proposed control sets could play a pivotal roll in future any-angle path planning.

Chapter 4

Tunable Trajectory Planner Using G^3 Curves

4.1 Introduction

In this chapter, we present a technique to compute trajectories between start and goal configurations. The trajectories we compute are C^3-continuous with respect to time implying that they are thrice continuously differentiable.

We focus on configurations of the form $(x, y, \theta, \kappa, \sigma, v, \alpha)$ where $(x, y) \in \mathbb{R}^2$ represents position in the plane, θ the heading, κ the curvature, σ the curvature rate, v the velocity, and α the velocity rate (all derivatives taken with respect to arc-length). In particular, we seek to compute trajectories that optimize a trade off between comfort and travel time [66, 121], though the methods developed here can be used to consider other features as well.

Comfort of a trajectory is related to the acceleration, the rate of change of yaw, and jerk (the derivative of the acceleration) experienced by a vehicle sliding along it [49, 121]. A common metric for comfort is the integral of the square of the jerk (IS jerk) over the trajectory [66]. When a vehicle is moving at constant speed, jerk is experienced lateral to the trajectory and is proportional to the derivative of curvature—called the *sharpness*—of the vehicle trajectory. It is important to note that the smoothness and comfort of the resulting vehicle motion depends on both the reference trajectory and the tracking controller. However, by planning trajectories with low jerk, less effort is required from the controller to track the reference and ensure comfort [64], particularly at high speeds [102].

Often, trajectory planning is treated as a two part spatio-temporal problem [57]. The first sub-problem deals with computing a path from start to goal, while the second addresses how this path should be converted into a motion by computing a velocity profile. Optimizing a travel time/comfort trade off over a set of paths and velocity profiles ensures the resulting trajectory's adherence to desirable properties like short travel time, and comfort.

The problem just described is an instance of infinite dimensional optimization in which a non-convex cost (discussed later) is minimized over two continuous functions: path and velocity profile. On one extreme, one could attempt to compute both functions simultaneously. However, because of the nature of the problem, conventional techniques like convex optimization [36], or sequential quadratic programming, cannot be used without first modifying the problem. On the other extreme, one could consider a simplified version of the problem by completely decoupling path and velocity: first minimizing cost over one function, then the other. This latter approach is widely used [100, 119], but does not consider the influence of the choice of path on the velocity profile (or vice-versa). For example, if a user strongly prefers short travel time to comfort, then an algorithm which first selects a cost-minimizing path, and then computes a velocity given that path would compute a trajectory with the shortest possible path, and highest feasible velocity. However, it has been observed that such a trajectory is sub-optimal when attempting to minimize travel time. This is because short paths typically feature higher values of curvature which require lower values of velocity to maintain safety.

We propose a hybrid of these two approaches: we decouple the problem but repeatedly solve the path and velocity profiles in an iterative fashion. In particular, we consider a simplification of the original optimization problem that limits the set of admissible paths to one in which individual elements can easily be distinguished from each other via a single parameter $\bar{\rho} \in \mathbb{R}_{>0}$. This is done by considering only those paths whose second derivative of curvature is piece-wise constant, taking only values in $\{0, \pm\bar{\rho}\}$. We also propose a modification of the techniques employed by [11] to discretize the set of admissible velocity profiles into $N+1 \geq 5$ way-points for longitudinal jerk. Thus, the two continuous functions, path and velocity profile, over which the our cost is minimized are replaced with $N+2$ constants. We then repeatedly select paths by selecting values for $\bar{\rho}$, and compute the $N+1$ remaining parameters that minimize cost given the selected path. We thus iteratively refine both path and velocity profile. For clarity, the contributions of this chapter are repeated here from Section 1.1.3.

1. Given a set of weights representing a trade off between comfort and travel time, we propose a method of simplifying the resulting infinite-dimensional non-convex optimization

problem to one of finite dimension, allowing us to iteratively refine both the path and velocity.

2. To use the technique proposed here, we develop a method to compute paths between start-goal configurations whose second derivative of curvature is piece-wise constant taking values only in $\{0, \pm\bar{\rho}\}$ given a value $\bar{\rho} \in \mathbb{R}_{>0}$.

3. Finally, we present a modification of the technique from [11] that allows us to compute a cost-minimizing velocity profile given a path.

4.2 Problem Statement

We begin with a review of the optimization problem that motivated the development of the Continuous Curvature Rate and Hybrid Curvature Rate paths presented in [6]. These are G^3 continuous paths in which the derivative of curvature with respect to arc-length σ is a piece-wise linear continuous function of arc-length and the second derivative of curvature with respect to arc-length ρ is piece-wise constant. We will then augment the problem to account for comfort resulting in a new problem that is the focus of this chapter. A note on notation: we use $(\cdot)'$ to denote differentiation with respect to arc-length s along a path, while $\dot{(\cdot)}$ represents differentiation with respect to time t.

4.2.1 Original Optimization Problem

Recall from the Introduction: A curve is called G^k-continuous, or *geometrically* continuous, if it is $k-$times continuously differentiable with respect to arc-length [7]. Typically, a path is said to be G^k if it is G^k continuous, but not G^{k+1} continuous.

The set of all G^3 curves in 2D space is the set of all solutions to the following differential equation [6]

$$\begin{bmatrix} x' \\ y' \\ \theta' \\ \kappa' \\ \sigma' \end{bmatrix} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ \kappa \\ \sigma \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \rho \end{bmatrix}, \tag{4.1}$$

where $(x, y) \in \mathbb{R}^2, \theta \in [0, 2\pi), \kappa \in \mathbb{R}, \sigma \in \mathbb{R}$ denote the instantaneous position, heading, curvature, and curvature rate along the curve, respectively. Note, σ is the derivative of

curvature with respect to arc-length (called the *sharpness*, or *curvature rate*), while ρ represents the second derivative of curvature with respect to arc-length. Observe that the model in (4.1) does not assume that ρ is continuous. Thus model (4.1) can be used to describe G^3 curves whose curvature rates are piece-wise linear functions of arc-length. The goal of path planning using G^3 curves is to obtain paths of minimal final arc-length s_f, from start configurations $(x_s, y_s, \theta_s, \kappa_s, \sigma_s)$ to a goal configurations $(x_g, y_g, \theta_g, \kappa_g, \sigma_g)$, subject to boundary constraints:

$$\begin{aligned} x(0) = x_s,\ y(0)\ = y_s,\ \ \theta(0)\ = \theta_s,\ \kappa(0),\ = \kappa_s, \\ x(s_f) = x_g,\ y(s_f) = y_g,\ \theta(s_f) = \theta_g,\ \kappa(s_f) = \kappa_g, \\ \sigma(0) = \sigma(s_f) = 0. \end{aligned} \tag{4.2}$$

and physical constraints:

$$\begin{aligned} \kappa(s) \in [-\kappa_{\max}, \kappa_{\max}],\ \forall s \in [0, s_f], \\ \sigma(s) \in [-\sigma_{\max}, \sigma_{\max}],\ \forall s \in [0, s_f], \\ \rho(s) \in [-\rho_{\max}, \rho_{\max}],\ \forall s \in [0, s_f], \end{aligned} \tag{4.3}$$

where $\kappa_{\max}, \sigma_{\max}, \rho_{\max} > 0$ are known. To summarize, if velocity is restricted to positive values, then the work of [6] seeks to solve the following optimization problem (OP):

$$\begin{aligned} \min_{\rho}\ s_f \\ s.t.\ \text{constraints } (4.1), (4.2), (4.3) \end{aligned} \tag{4.4}$$

Observe that at constant speed, minimizing arc-length as in (4.4), is equivalent to minimizing travel time.

4.2.2 Adding Comfort Constraints & Velocity

The constraints (4.3) involve the positive constants $\kappa_{\max}$, $\sigma_{\max}$, $\rho_{\max}$. As in [29] and [6] we assume that these values are known and reflect maximum physical limits of a vehicle. That is, $\kappa_{\max}^{-1}$ is the minimum turning radius of the vehicle, $\sigma_{\max}$ is the maximum curvature rate given limitations on the vehicle's steering actuator, etc. Observe that by (4.1), any function ρ uniquely defines path, denoted π_ρ from a fixed start. For the purposes of this chapter, we assume known initial and final velocities v_s, v_g, (resp.) and zero initial and final acceleration. This ensures smooth concatenation of trajectories, and is common in trajectory generation [46]. Under these assumption, a continuous velocity profile is uniquely determined by v'', denoted β.

The functions ρ and β will serve as the variables of an OP that balances travel time and discomfort. Any choice of (ρ, β) uniquely defines a trajectory given by the tuple (π_ρ, v_β) where π_ρ is a feasible solution to (4.4), and v_β is the velocity profile associated with β. If ρ and β are parameterized by arc-length, we write $(\pi_\rho(s), v_\beta(s))$, whereas we write $(\pi_\rho(t), v_\beta(t))$ if ρ and β are parameterized by time.

Observe that computing a velocity profile for a G^3 curve π_ρ induces a re-parametrization. Further, by (4.1) the third derivative of position with respect to the new parametrization variable t is a C^∞ function of $\theta, \kappa, \sigma, v, v', \beta$. Therefore, in order for the trajectory with path π_ρ and velocity profile v to be a thrice continuously differentiable (w.r.t. t), we require that β be continuous functions of t.

Using a technique similar to [66, 67, 121, 122], we penalize a trade off between travel time and comfort:

$$\begin{aligned} C(\pi_\rho(t), v_\beta(t)) &= \int_0^{t_f} (w_a C_a + w_{\mathcal{J}} C_{\mathcal{J}} + w_y C_y + w_t) dt, \\ C_a = |a_N(t)|^2 + |a_T(t)|^2, C_y &= \left| \frac{d\theta(t)}{dt} \right|^2, \\ C_{\mathcal{J}} &= \mathcal{J}_N(t)^2 + \mathcal{J}_T(t)^2. \end{aligned} \tag{4.5}$$

Here, $C_a, C_{\mathcal{J}}, C_y$ represent the squared magnitude of the acceleration $\boldsymbol{a} = \dot{v}$ (expressed using normal and tangential components a_N, a_T), the squared magnitude of jerk (expressed using normal and tangential components $\mathcal{J}_N, \mathcal{J}_T$), and the squared magnitude of yaw rate for the trajectory $(\pi_\rho(t), v_\beta(t))$. These costs are weighted with constants $w_a, w_{\mathcal{J}}, w_y$ representing the relative importance of each feature to a user. We refer to the terms $\int_0^{t_f} w_m C_m, m \in \{a, \mathcal{J}, y, t\}$ as the integral squared (IS) acceleration, IS jerk, IS yaw, and time cost, respectively.

A method similar to optimize the velocity profile v, for a given fixed path is provided later. In order to use this method, we require that the limits of integration in the definition of the cost be fixed. Noting that the final arc-length of a given path $\pi_\rho(t)$ is fixed, we re-parameterize the cost (4.5) in terms of arc-length s. Letting $\boldsymbol{n}, \boldsymbol{\tau}$ represent the unit normal and unit tangent vectors, respectively, we observe that the acceleration vector $\boldsymbol{a}$, and the jerk vector $\boldsymbol{\mathcal{J}}$ are given by

$$\begin{aligned} \boldsymbol{a} &= a_N \boldsymbol{n} + a_T \boldsymbol{\tau} = |\kappa| v^2 \boldsymbol{n} + a_T \boldsymbol{\tau} \\ \boldsymbol{\mathcal{J}} &= \dot{\boldsymbol{a}} = (3 v a_T |\kappa| + v^3 \bar{\sigma}) \boldsymbol{n} + (\dot{a}_T - \kappa^2 v^3) \boldsymbol{\tau}, \end{aligned} \tag{4.6}$$

where $\bar{\sigma} = (|\kappa|)'$. The expression for $\boldsymbol{\mathcal{J}}$ was obtained by differentiating $\boldsymbol{a}$ with respect to time (using the Frenet-Serret formula to integrate the normal and tangent vectors, see [101])

and observing $\dot{m} = m'(ds/dt) = m'v$ for any state m. Finally, letting $\alpha = v', \beta = \alpha', b = \dot{a}_T$, we observe

$$a_T = \alpha v, \;\; \dot{a}_T = b = v(\beta v + \alpha^2). \tag{4.7}$$

Combining (4.5), (4.6), and (4.7), and integrating with respect to s instead of t yields:

$$\begin{aligned} C(\pi_\rho(s), v_\beta(s)) &= \int_0^{s_f} w_a \tilde{C}_a + w_{\mathcal{J}} \tilde{C}_{\mathcal{J}} + w_y \tilde{C}_y + w_t \tilde{C}_t ds, \\ \tilde{C}_a &= v^3 |\kappa|^2 + \alpha^2 v, \\ \tilde{C}_{\mathcal{J}} &= v^3 (3\alpha|\kappa| + \bar{\sigma} v)^2 + v(\beta v + \alpha^2 - \kappa^2 v^2)^2, \\ \tilde{C}_y &= |\kappa|^2 v, \\ \tilde{C}_t &= v^{-1}. \end{aligned} \tag{4.8}$$

We now use this cost together with the constraints developed earlier to state the new OP that is the focus of this chapter:

$$\begin{aligned} \min_{\rho(s), \beta(s)} \;\; & C(\pi_\rho(s), v_\beta(s)) \\ s.t. \;\; & \text{constraints } (4.1), (4.2), (4.3), (4.7) \\ & \begin{bmatrix} v'_\beta(s) & \alpha'(s) \end{bmatrix}^T = \begin{bmatrix} \alpha(s) & \beta(s) \end{bmatrix}^T \\ & v_\beta(0) = v_s, \;\; v_\beta(s_f) = v_f, \;\; \alpha(0) = \alpha(s_f) = 0, \\ & v_\beta(s) \in (0, v_{\max}], a(s) \in [-a_{\max}, a_{\max}], \\ & b(s) \in [-b_{\max}, b_{\max}], \;\; \forall s \in [0, s_f]. \end{aligned} \tag{4.9}$$

Let $R \times B$ be the set of all $(\rho(s), \beta(s))$ whose associated trajectories are feasible solutions to (4.9). In the next section, we describe our solution approach in detail. Though this work focuses on optimizing a trade-off between travel time and comfort, the techniques developed herein can be extended to account for other trajectory features as well. This can be accomplished by simply adding and/or removing features in the cost function (4.9). The techniques developed here require only that the integrand of the cost be polynomial in $v(s), \alpha(s), \beta(s)$ and not explicitly contain ρ (i.e., $\partial C / \partial \rho = 0$).

4.3 Approach

The optimization problem in (4.9) is an instance of infinite dimensional, non-convex optimization. The non-convexity of (4.9) is due to the cost $C_{\mathcal{J}}$ (and $\tilde{C}_{\mathcal{J}}$). Furthermore, the

non-holonomic constraints (4.1) require the evaluation cubic Fresnel integrals for which there is no closed form solution. For these reasons, we propose an approach that simplifies (4.9). In this section we present the high-level idea behind our technique, beginning with a motivating Theorem.

Theorem 4.3.1 (Optimal G^3 paths)**.** *If (ρ^*, β^*) is a solution to the OP (4.9), then*

1. *The function $\rho^*(s)$ is piece-wise constant.*
2. *The optimal path $\pi_{\rho^*}(s)$ is G^3-continuous.*
3. *The previous two results persist if the cost function in (4.9) is replaced with any other cost function C' provided that $\partial C'/\partial \rho = 0$.*
4. *Even if the magnitude constraint $\rho(s) \in [-\rho_{max}, \rho_{max}], \forall s \in [0, s_f]$ is removed from (4.9), the path of the optimal trajectory will be G^3-continuous.*

With the constraint $\rho(s) \in [-\rho_{\max}, \rho_{\max}], \forall s \in [0, s_f]$ in place, the first result of this Theorem follows directly from the observation that ρ does not appear explicitly in the cost of (4.9), and appears linearly in the constraints. Thus the Hamiltonian is linear in ρ, and the first result follows from Pontryagin's Minimum Principle [53, Chapter 12]. Since the $\sigma^*(s)$ – the function $\sigma(s)$ for the optimal path – is continuous but $\rho^*(s)$ is not, we can conclude that $\pi_{\rho^*}(s)$ is G^3-continuous and the second result holds. Observing that the above proof can be applied to any cost function with $\partial C'/\partial \rho = 0$ implies the third result. The final result of this Theorem follows by observing that $\sigma(s)^*$ – the function $\sigma(s)$ for the optimal path – must be continuous even if the constraint $\rho(s) \in [-\rho_{\max}, \rho_{\max}]$ is removed from (4.9). Indeed, minimizing the Hamiltonian with respect to σ (by Pontryagin's Minimum Principle), we see that the result is a continuous function of arc-length by the Weierstrass-Erdmann corner conditions [68, Section 3.1.1].

Theorem 4.3.1 implies that a path that solves (4.9) is a G^3 path with possibly discontinuous, piece-wise constant function $\rho(s)$. However, there are many such paths connecting start and goal path states. Therefore, the first part of our technique is to simplify the OP (4.9) by considering only those G^3 curves $\pi_{\hat{\rho}}$ such that there exists a constant $\bar{\rho} \leq \rho_{\max}$ for which $\pi_{\hat{\rho}}$ solves (4.4) when $\rho_{\max}$ is replaced with $\bar{\rho}$. That is, we approximate a solution to (4.9) by solving

$$\begin{aligned} &\min_{\bar{\rho} \leq \rho_{\max}} \left(\min_{\beta(s) \in B} C(\pi_{\hat{\rho}}(s), v_\beta(s)) \right), \\ &s.t., \hat{\rho}(s) \in R, \text{ solves (4.4) for } \rho_{\max} = \bar{\rho}. \end{aligned} \tag{4.10}$$

In words, (4.10) approximates the OP in (4.9) by replacing the continuous function $\rho(s)$ with a constant $\bar{\rho}$. The path associated with $\bar{\rho}$ is a shortest G^3 path $\pi_{\hat{\rho}}(s)$ such that $\hat{\rho}(s)$ is bounded in magnitude by $\bar{\rho}$. From the results in [6], we observe that $\hat{\rho}$ is a function taking values only in $\{\pm\bar{\rho}, 0\}$. This approach has two major advantages: first, we have replaced the continuous function $\rho(s)$ in (4.9) with constant $\bar{\rho}$ while still maintaining the optimal form of the G^3 path (piece-wise constant in $\hat{\rho}$). Second, by tuning $\bar{\rho}$, we can still produce G^3 curves with both sharp (typically favored by users who value short travel times) and gradual curvature functions (favored by users who value comfortable trajectories).

The second part of our technique involves simplifying (4.10) yet further by replacing the continuous function $\beta(s)$ with a decision vector. Similar to [11], we discretize the arc-length along the path $\pi_{\hat{\rho}}$ into $N+1 \in \mathbb{N}_{\geq 5}$ fixed points $\{s_0, ..., s_N\}$. We then replace the continuous function $\beta(s)$ with the decision vector $\boldsymbol{B} = [b_i, i = 0, \ldots, N]$ where b_i is the longitudinal jerk at arc-length s_i. These way-points b_i can then be used to produce a continuous, piece-wise linear function $\beta(s)$ by connecting sequential values b_i, b_{i+1} with straight lines. In our implementation we use $N = 10$, which works well in practice.

Thus far, our approach has simplified (4.9) by replacing the two continuous functions ρ, β with $N+2$ constants $\bar{\rho}, b_i, i = 0, \ldots, N$. The final stage of our technique is to compute the cost-minimizing values of these constants. The high level idea is to iteratively select values of $\bar{\rho}$, compute the associated path $\pi_{\hat{\rho}}(s)$ for each selection, and then compute the cost minimizing vector $\boldsymbol{B}$ given the fixed path. For each $\bar{\rho}$, the decision vector $\boldsymbol{B}$ can be efficiently computed as the integrand of the cost C in (4.10) is a polynomial expression of v, α, β for a fixed path, facilitating the use of second order optimization techniques (Section 4.5).

Unfortunately, the same techniques can not be applied to compute $\bar{\rho}$. Even infinitesimal changes to the value $\bar{\rho}$ can result in large changes to the curvature and curvature rate profiles of the resulting path as well as the final arc-length. The resulting change in cost is then exacerbated when a velocity profile is computed that responds to the new curvature and curvature rate. This adversely affects the calculation of first and second derivatives of cost with respect to $\bar{\rho}$, hindering the use of derivative-based optimization techniques. The phenomenon is analogous to the case of Dubins' paths when infinitesimally changing the maximum curvature completely alters the structure of the path from, say, two curves connected by a straight line to three curves. For this reason, we appeal to sampling-based, derivative-free optimization techniques to compute the optimal value of $\bar{\rho}$.

In order to use the methods described above, we require two sub-techniques: The first solves (4.4) for any given values $\sigma_{\max}, \kappa_{\max}$, and $\rho_{\max} = \bar{\rho}$ (Section 4.4). The second computes the optimal way-points $\alpha(s_i)$ for any fixed path (Section 4.5).

4.4 Computing G^3 Paths

In this section, we describe how to compute G^3 paths that solve (4.4) for known values of $\rho_{\max}$. The solution $\rho(s)$ to (4.4) is piece-wise constant, taking values only in $\{0, \pm\rho_{\max}\}$ [6], a direct result of Pontryagin's Minimum Principle [53, Chapter 12]. We begin with an investigation of single G^3 curves, following closely the work presented in [6]. However, in [6], it is assumed that the initial and final curvatures of all paths are either 0 or maximum in magnitude. Therefore, it is necessary to re-derive all relevant equations to remove this assumption. We then present a technique to connect G^3 curves to form G^3 paths.

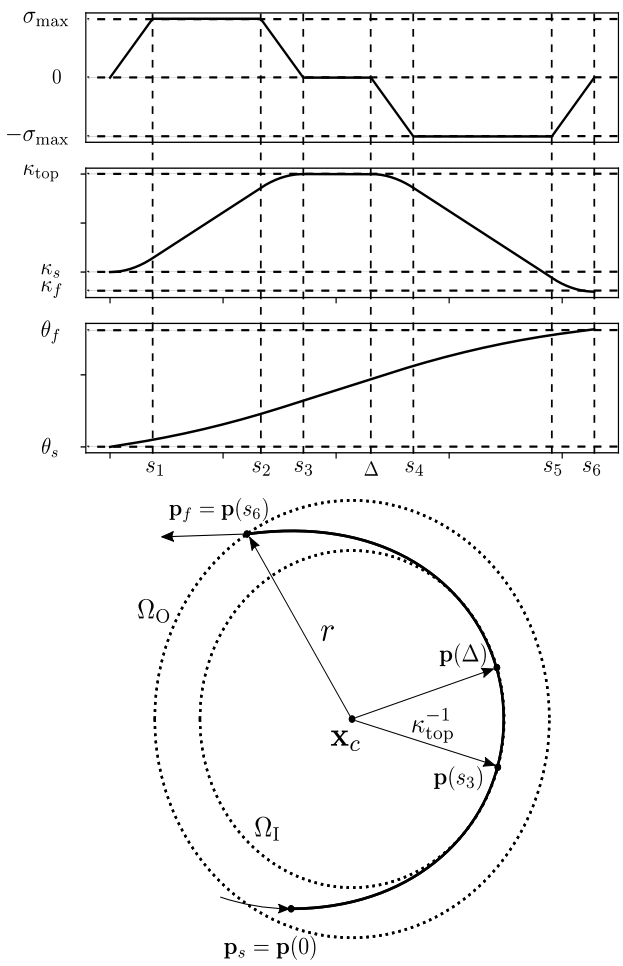


Figure 4.1: A basic G^3 curve. **Top**: The functions $\sigma(s)$ (top), $\kappa(s)$ (mid), and $\theta(s)$ (bottom). **Bottom**: The resulting curve in the x, y plane from start configuration $\boldsymbol{p}_s$ to final configuration $\boldsymbol{p}_f$. Image also appears in [6]

4.4.1 Single G^3 Curves

This section uses the definitions and notation presented in [6] to outline the general form of a G^3 curve.

An example G^3 curve performing a left-hand turn is presented in Figure 4.1. The maximum curvature of this maneuver is κ_{top} where $|\kappa_{\text{top}}| \leq \kappa_{\max}$. The curve begins at a path state $\boldsymbol{p}_s = (x_s, y_s, \theta_s, \kappa_s, \sigma_s = 0)$ at an arc-length of $s = 0$ along the path. From $s = 0$

to $s = s_1$, the second derivative of curvature ρ, is set to its maximum value $\rho_{\max}$. Thus the curvature rate σ is given by the function $\sigma(s) = \rho_{\max} \cdot s$. The functions $\kappa(s), \theta(s)$ are therefore quadratic and cubic (resp.), and can be calculated by (4.1).

At $s = s_1$, the curvature rate has reached its maximum allowable value $\sigma_{\max}$, and can go no higher, thus the function ρ is set to 0, and $\sigma(s) = \sigma_{\max}$. As a result, for $s_1 \leq s \leq s_2$, the function $\kappa(s), \theta(s)$ are linear and quadratic (resp.). At $s = s_2$, the value of ρ is set to $-\rho_{\max}$, and the curvature rate decreases from $\sigma_{\max}$, allowing the curvature to reach a value κ_{top} at $s = s_3$. At $s = s_3$, the path state is given by $\boldsymbol{p}(s_3)$. From $s = s_3$ to $s = \Delta$, the curvature of the curve is held constant at κ_{top}. Thus the curve remains at a constant distance κ_{top}^{-1} from its center of curvature $\boldsymbol{x}_c$ for all $s_3 \leq s \leq \Delta$. The circle Ω_{I} is the circle centered at $\boldsymbol{x}_c$ with radius κ_{top}^{-1}.

At $s = \Delta$, the value of ρ is set to $-\rho_{\max}$, and the curvature rate follows the linear function $\sigma(s) = -\rho_{\max}(s - \Delta)$ until $s = s_4$ when $\sigma(s) = -\sigma_{\max}$. Upon reaching its minimum allowable value, $\sigma(s)$ remains constant at $-\sigma_{\max}$ for $s_4 \leq s \leq s_5$. At $s = s_5$, the value of ρ is set to $\rho_{\max}$, allowing the curvature to descend to its final value κ_f at $s = s_6$. When $s = s_6$, the path state is given by $\boldsymbol{p}(s_6)$ which lies a distance of r away from $\boldsymbol{x}_c$. The circle Ω_{O} is centered at $\boldsymbol{x}_c$ with radius r. We use a subscript f to represent a path state at the end of a G^3 curve. This is to differentiate these states from ones with a subscript g which represents the goal path states at the end of a G^3 path (the concatenation of one or more curves with straight lines).

To perform a right-hand turn, a similar analysis to that above is employed. The resulting function $\sigma(s)$ will appear as a mirror images about the horizontal axis to that presented in Figure 4.1. We now illustrate how the switching arc-lengths $s_i, i = 1, \ldots, 6$ of the function $\rho(s)$ may be calculated. While similar, the switching arc-lengths presented in [6] assume that $\kappa_s, \kappa_f \in \{0, \pm\kappa_{\max}\}$. Because this is not an assumption that we make, it is necessary to re-derive these values.

Given the parameters $\rho_{\max}, \sigma_{\max}, \kappa_{\text{top}}, \Delta$, the values of $s_i,\ i = 1, \ldots, 6$ such that $\kappa(0) =$

$\kappa_s, \kappa(s_3) = \kappa_{\text{top}}, \kappa(s_6) = \kappa_f$, are given by:

$$
\begin{aligned}
s_1 &= \begin{cases} \frac{\sigma_{\max}}{\rho_{\max}}, & \text{if } |\kappa_{\text{top}} - \kappa_s| > \frac{(\sigma_{\max})^2}{\rho_{\max}} \\ \sqrt{\frac{|\kappa_{\text{top}} - \kappa_s|}{\rho_{\max}}}, & \text{otherwise} \end{cases} \\
s_2 &= \begin{cases} \frac{|\kappa_{\text{top}} - \kappa_s|}{\sigma_{\max}}, & \text{if } |\kappa_{\text{top}} - \kappa_s| > \frac{(\sigma_{\max})^2}{\rho_{\max}} \\ s_1, & \text{otherwise} \end{cases} \\
s_3 &= s_1 + s_2, \\
s_4 &= \Delta + \begin{cases} \frac{\sigma_{\max}}{\rho_{\max}}, & \text{if } |\kappa_{\text{top}} - \kappa_f| > \frac{(\sigma_{\max})^2}{\rho_{\max}} \\ \sqrt{\frac{|\kappa_{\text{top}} - \kappa_f|}{\rho_{\max}}}, & \text{otherwise} \end{cases} \\
s_5 &= \Delta + \begin{cases} \frac{|\kappa_{\text{top}} - \kappa_f|}{\sigma_{\max}}, & \text{if } |\kappa_{\text{top}} - \kappa_f| > \frac{(\sigma_{\max})^2}{\rho_{\max}} \\ s_4, & \text{otherwise} \end{cases} \\
s_6 &= s_4 + s_5 - \Delta.
\end{aligned}
\tag{4.11}
$$

Also similar to [6], the curvature function of a G^3 curve with $\rho(s)$ switching arc-lengths given by (4.11) and with with initial curvature κ_s, is given by

$$
\kappa(s) = \begin{cases}
\kappa_s \pm 0.5\rho_{\max} s^2, \ s \in [0, s_1] \\
\kappa(s_1) \pm \rho_{\max} s_1 (s - s_1), \ s \in [s_1, s_2] \\
\kappa(s_2) \pm 0.5\rho_{\max} (s - s_2)(s_2 + 2s_1 - s), \ s \in [s_2, s_3] \\
\kappa_{\text{top}}, \ s \in [s_3, \Delta] \\
\kappa_{\text{top}} \mp 0.5\eta\rho_{\max} (s - \Delta)^2, \ s \in [\Delta, s_4] \\
\kappa(s_4) \mp \eta\rho_{\max} (s_4 - \Delta)(s - s_4), \ s \in [s_4, s_5] \\
\kappa(s_5) \mp 0.5\eta\rho_{\max} (s - s_5)(s_5 - 2\Delta + 2s_4 - s).
\end{cases}
\tag{4.12}
$$

Here, the top sign of each "$\pm, \mp$" is used if $\kappa_{\text{top}} \geq \kappa_s$, while the bottom sign is used otherwise and $\eta = 1$ if $\sigma(s_1)\sigma(s_4) < 0$ and $\eta = -1$ otherwise. The value $\eta = 1$ is used in circumstances where the curvature function increases to κ_{top} and then decreases to κ_f, or decreases to κ_{top} and then increases to κ_f. On the other hand, a value $\eta = -1$ is used when the curvature function is either monotonically increasing or decreasing over the entire curve.

By (4.1), the path state vector $\boldsymbol{p}$ along a G^3 curve whose curvature is given by (4.12),

can be parameterized in terms of arc-length as:

$$\boldsymbol{p}(s) = \begin{bmatrix} x(s) \\ y(s) \\ \theta(s) \\ \kappa(s) \end{bmatrix} = \begin{bmatrix} x_s + \int_0^s \cos(\theta(\tau))d\tau \\ y_s + \int_0^s \sin(\theta(\tau))d\tau \\ z_0 + z_1 s + z_2 s^2 + z_3 s^3 \\ \kappa(s) \end{bmatrix}, \tag{4.13}$$

where z_0, z_1, z_2, z_3 are obtained by integrating $\kappa(s)$ in (4.12). As Figure 4.1 implies, the point $\boldsymbol{x}_c$, and radius r, are given by

$$\boldsymbol{x}_c = \begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} x(s_3) - \kappa_{\text{top}}^{-1} \sin(\theta(s_3)) \\ y(s_3) + \kappa_{\text{top}}^{-1} \cos(\theta(s_3)) \end{bmatrix}, \tag{4.14}$$

$$r = ||(x(s_6), y(s_6)) - (x_c, y_c)||. \tag{4.15}$$

Assuming that $\rho_{\max}, \sigma_{\max}$ are known, we observe from the preceding arguments that the initial path states $\boldsymbol{p}_s$, the top and final curvatures $\kappa_{\text{top}}, \kappa_f$ (resp.), and the arc-length Δ are enough to uniquely define a G^3 curve. Thus we denote a G^3 curve $G(\boldsymbol{p}_s, \kappa_{\text{top}}, \kappa_f, \Delta)$ as the set of path states $\boldsymbol{p}(s)$ given by (4.13) from $s = 0$ to $s = s_6$. A G^3 path π_ρ, solving (4.4) is therefore a concatenation of such curves and straight lines where $\rho(s)$ is a function that takes only values in $\{\pm\rho_{\max}, 0\}$.

As (4.11) implies, the final arc-length of this curve is given by s_6. Let $G_i^{s_j}(\boldsymbol{p}_s, \kappa_{\text{top}}, \kappa_f, \Delta)$, be the value of path state i at arc-length s_j. In the next section, we present our technique for solving (4.4) by concatenating G^3 curves and straight lines.

4.4.2 Connecting G^3 Curves

This section proposes a technique to connect G^3 curves using a straight line via a reduction of the G^3 curve path planning problem (4.4) to a Dubin's-like path planning problem with two different minimal turning radii. The latter problem can be solved quickly by calculating the common tangents of two circles with different radii. For fixed values $\boldsymbol{p}_s, \kappa_{\text{top}}, \kappa_f = 0$, consider the set of curves

$$\mathcal{G} = \{G(\boldsymbol{p}_s, \kappa_{\text{top}}, \kappa_f = 0, \Delta) \ : \ \Delta \geq s_3\}. \tag{4.16}$$

Observe that members of $\mathcal{G}$ are G^3 curves with final curvature 0. This is to facilitate connecting G^3 curves using straight lines while preserving curvature continuity. The assumption that $\sigma(s_6) = 0$ ensures that G^3 curves can be connected with straight lines while preserving continuous curvature rate.

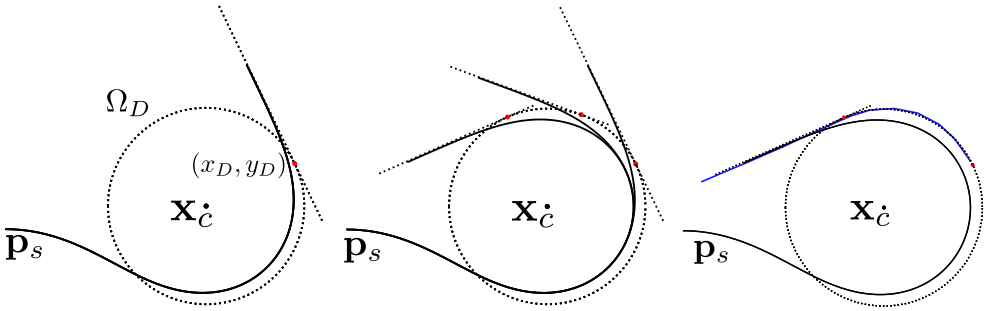


Figure 4.2: (Left) Representative circle Ω_D, and point (x_D, y_D). (Mid) Illustration of Theorem 4.4.2. (Right) Illustration of Step 5, with partial Dubin's path $\mathcal{D}$ (blue) and corresponding G^3 curve $\hat{G}_i$ (black).

For each pair $(\boldsymbol{p}_s, \kappa_{\text{top}})$ we can construct a set $\mathcal{G}$ in (4.16), containing the curve $G(\boldsymbol{p}_s, \kappa_{\text{top}}, \kappa_f = 0, \Delta = s_3)$ as an element. Upon reaching curvature κ_{top}, the curvature of this curve immediately increases or decreases to κ_f. The curvature profile of such a curve can be found in Figure B.1.

For fixed values $\boldsymbol{p}_s, \kappa_{\text{top}}, \kappa_f = 0$, the value of s_3 given in (4.11) is independent of Δ, implying that every curve in $\mathcal{G}$ shares the same switching arc-length s_3. The same holds for the center $\boldsymbol{x}_c$ given in (4.14). Therefore, the values $\boldsymbol{p}_s, \kappa_{\text{top}}, \kappa_f = 0$, which are shared by all curves in $\mathcal{G}$, are sufficient to determine s_3 and $\boldsymbol{x}_c$. Finally, note that each curve in $\mathcal{G}$ is coincident with every other curve in $\mathcal{G}$ for all $s \leq s_3$. This phenomena can be see in Figure 4.1: the duration $\Delta - s_3$ that a curve spends at constant curvature κ_{top} does not affect the portion of the curve between $\boldsymbol{p}_s$ and $\boldsymbol{p}(s_3)$. The following Lemma illustrates the relationship between Δ and $G_\theta^{s_6}(\boldsymbol{p}_s, \kappa_{\text{top}}, \kappa_f, \Delta)$.

Lemma 4.4.1 (Final Heading)**.** *For each set of curves $\mathcal{G}$ defined in 4.16, there is a unique solution Δ to the equation $G_\theta^{s_6}(\boldsymbol{p}_s, \kappa_{top}, \kappa_f, \Delta) = \theta_f$ such that $G(\boldsymbol{p}_s, \kappa_{top}, \kappa_f, \Delta)$ has minimum arc-length over Δ:*

$$\Delta = s_3 + (\theta_g - G_\theta^{s_6}(\boldsymbol{p}_s, \kappa_{top}, \kappa_f, \Delta = s_3))\kappa_{top}^{-1}. \tag{4.17}$$

where s_3 is given by (4.11) for the curve $G(\boldsymbol{p}_s, \kappa_{top}, \kappa_f, \Delta)$.

The proof of Lemma 4.4.1 can be found in Appendix B. Lemma 4.4.1 implies that the final headings of elements in $\mathcal{G}$ varies linearly with their values of Δ. This Lemma motivates the following key Theorem that will be heavily leveraged later.

Theorem 4.4.2 (Reducing Theorem)**.** *Given a set of curves $\mathcal{G}$ in (4.16), let x_f, y_f, θ_f denote the final x, y, and θ values of any curve $G(\boldsymbol{p}_s, \kappa_{top}, 0, \Delta) \in \mathcal{G}$. Also, let m denote the slope of a line induced by θ_f, i.e., $m = \tan\theta_f$. The shortest distance from the point $\boldsymbol{x}_c$ defined in (4.14) to the line passing through (x_f, y_f) whose slope is m is a constant with respect to Δ.*

The proof of Theorem 4.4.2 can be found in Appendix B. The result of Theorem 4.4.2 is shown in Figure 4.2 (Mid). In this figure, the red dots correspond to the points on the lines containing the final path states of each curve in $\mathcal{G}$ that are closest to $\boldsymbol{x}_c$. Observe that they lie on a circle centered at $\boldsymbol{x}_c$. This theorem has the following major consequence that allows us to reduce a G^3 path planning problem to a Dubin's-like path planning problem if $\boldsymbol{p}_s, \boldsymbol{p}_g, \kappa_{\text{top}}, \sigma_{\max}, \rho_{\max}$ are fixed, and only Δ may vary. This interim result is the foundation of our proposed G^3 path planning technique. We begin with a definition.

Definition 4.4.3 (Representative Circle)**.** Given a set of curves $\mathcal{G}$ in (4.16), and using the same notation as Theorem 4.4.2, the *representative circle* Ω_D of a curve in $\mathcal{G}$, is the circle centered at $\boldsymbol{x_c} = (x_c, y_c)$ containing the point (x_D, y_D) where

$$\begin{aligned} x_D &= \begin{cases} \frac{(mx_f - y_f + m^{-1}x_c + y_c)}{m + m^{-1}}, & \text{if } \theta_f \neq \pi/2 \\ x_f, & \text{otherwise} \end{cases} \\ y_D &= \begin{cases} -m^{-1}(x_D - x_c) + y_c, & \text{if } \theta_f \neq \pi/2 \\ y_c & \text{otherwise.} \end{cases} \end{aligned} \tag{4.18}$$

That is, for the straight line containing (x_f, y_f) whose slope is $\tan\theta_f$, the representative circle is the circle centered at $\boldsymbol{x}_c$ containing the point on the line closest to $\boldsymbol{x}_c$. Observe that Theorem 4.4.2 implies that all curves in a set $\mathcal{G}$ have coincident representative circles. An example representative circle, and corresponding points (x_D, y_D) is show in Figure 4.2 (Left). In this figure, the black curve is $G(\boldsymbol{p}_s, \kappa_{\text{top}}, \kappa_f = 0, \Delta = s_3)$. Theorem 4.4.2 implies that every G^3 curve in a given set $\mathcal{G}$ in (4.16) has the same representative circle. This fact is leveraged in the following section.

4.4.3 Connecting G^3 curves with a Straight Line

We are now prepared to present our algorithm for computing G^3 paths that solve (4.4) given a known value of $\rho_{\max}$. The high-level idea is to begin by constructing two G^3 curves originating from both start configuration and goal configuration (with reverse orientation), and then to connect these curves using either a straight line or another curve. For simplicity,

let $\boldsymbol{p}_1 = \boldsymbol{p}_s, \boldsymbol{p}_2 = \boldsymbol{p}_g$, and let $\boldsymbol{p}_3$ denote $\boldsymbol{p}_g$ with reverse orientation. That is, $\boldsymbol{p}_3 = (x_g, y_g, \theta_g + \pi, -\kappa_g)$.

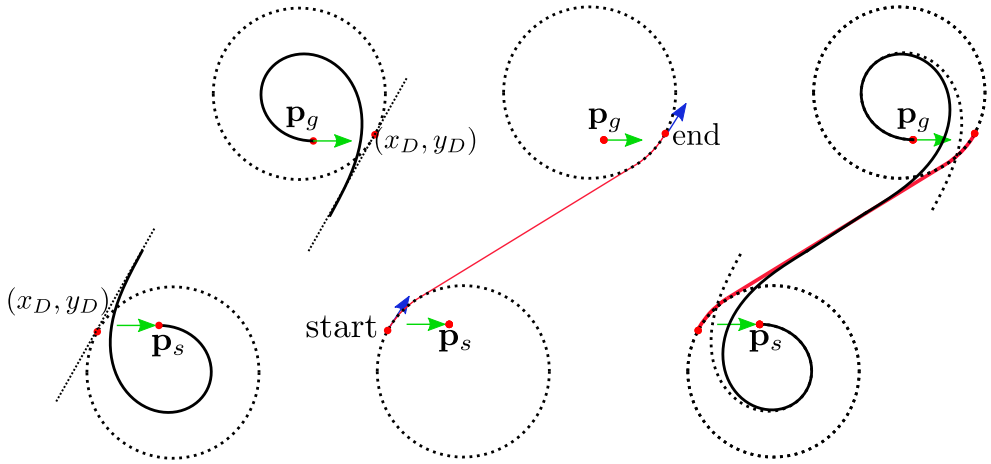


Figure 4.3: (Left) Representative circles around $\boldsymbol{p}_s$ and $\boldsymbol{p}_g$ in reverse. (Mid) Dubins'-like solution between start and goal states. (Right) Solution path.

Step 1: For each of the possible start and goal curve orientations: (R,L), (R,R), (L,R), (L, L) where R is a right-hand turn and L a left, perform the following steps, and then discard all but the choice with minimum final arc-length.

Step 2: Let $G_i = G(\boldsymbol{p}_i, |\kappa_{\text{top}_i}| = \kappa_{\max}, \kappa_{f_i} = 0, \Delta_i = s_{3_i}), i = 1, 3$ be two curves originating from $\boldsymbol{p}_1, \boldsymbol{p}_3$, respectively, whose top curvature reaches its maximum allowable magnitude $\kappa_{\max}$ (with sign given by the chosen orientation), and where s_{3_i} is given by (4.11). This step is illustrated by the black curves in Figure 4.3 (Left) for orientation (R,L). Let $\mathcal{G}_i$ denote the two sets of curves from (4.16) with $G_i \in \mathcal{G}_i$.

Step 3: Compute the points $(x_{D_i}, y_{D_i}), i = 1, 3$ for curves G_i on the representative circles of each curve. This step is illustrated in Figure 4.3 (Left). Here, the dotted circles represent the representative circles of each curve, and the red dots on these circles represent the points $(x_{D_i}, y_{D_i}), i = 1, 3$.

Step 4: Compute the Dubin's-type path $\mathcal{D}$ that consists of two curves and a straight line connecting $(x_{D_i}, y_{D_i}), i = 1, 3$ with minimum turning radius on each curve $r_{D_i} = ||(x_{D_i}, y_{D_i}) - (x_{c_i}, y_{c_i})||$. This step is illustrated in Figure 4.3 (mid). Note that curves

$G_i, i = 1, 3$ may have different values of r_{D_i}, resulting in a Dubin's problem with different minimum turning radii. This may not have a solution of the form described here, which will be addressed this in the next section.

Step 5: Compute the angle θ_f inscribed by the straight portion of $\mathcal{D}$ and the horizontal axis. Compute the value of $\hat{\Delta}_i$ from (4.17) such that the curves $\hat{G}_i = G(\boldsymbol{p}_i, \kappa_{\text{top}_i}, \kappa_{f_i}, \hat{\Delta}_i) \in \mathcal{G}_i$ $i = 1, 3$ have final headings $\theta_f, \theta_f + \pi$, respectively. By Theorem 4.4.2, curves $G, \hat{G}_i$ have coincident representative circles which, by construction, have radii r_{D_i} the minimum turning radii of $\mathcal{D}$. Therefore, the final states $(x_{f_i}, y_{f_i}, \theta_{f_i}, \kappa_i = 0, \sigma_{f_i} = 0)$ of the curves $\hat{G}_i$ are on the path $\mathcal{D}$. This is illustrated in Figure 4.2 (Right). Connect the curves $\hat{G}_i = G(\boldsymbol{p}_i, \kappa_{\text{top}_i}, \kappa_{f_i} = 0, \Delta_i)$ with a straight line. This step is illustrated in Figure 4.3 (Right). Solid black, and dotted black lines represent $\hat{G}_i, G_i$, respectively.

Step 6: (Looping) If $\sigma_{\max}, \rho_{\max}$ are too small compared to $\kappa_{\max}$, then one or both of the curves G_i in Step 2 will *loop* [6]. A curve $G(\boldsymbol{p}_s, \kappa_{\text{top}}, \kappa_f, \Delta)$ with initial and final headings θ_s, θ_f (resp.), will loop if $|\theta_f - \theta_s| < G_\theta^{s_6}(\boldsymbol{p}_s, \kappa_{\text{top}}, \kappa_f, \Delta = s_3)$. If such looping occurs on curve $i = 1, 3$, we decrease the magnitude of the top curvature κ_{top_i}, and return to Step 2. Minimum arc-length is guaranteed by computing the smallest magnitude values of κ_{top_i} that ensure a connection in Step 5.

4.4.4 Connecting G^3 curves with a G^3 curve

Two problems may arise in Steps 1-6 above. The first, we call *overlap*. Overlap arises when the endpoints of the two curves $\hat{G}_i, i = 1, 3$ computed in Step 5 cannot be connected with a straight line that preserves orientation. The phenomenon is illustrated in Figure 4.4 (Left), and can occur if the centers $\boldsymbol{x}_{c_i}$ of the representative circles of the curves G_i in Step 2 are too close together ($< r_1 + r_3$ where r_i is the radius (4.15) for the curve $\hat{G}_i$). Observe that overlap is a result of $\sigma_{\max}, \rho_{\max}$ being small relative to κ_{top_i}: by the time the curvature of curve $\hat{G}_i$ changes from κ_{top_i} to 0, the final configuration has already passed that of the second curve and cannot be connected with a straight line that preserves orientation. Therefore, if overlap occurs we attempt to find a curvature other than 0 that can be used to connect $\hat{G}_i$. This is summarized here:

Step 7: (Overlap) If overlap occurs, we cut the curves $\hat{G}_i, i = 1, 3$ at $s = \Delta_i$. That is, let $\hat{G}_1^{\text{cut}} = G(\boldsymbol{p}_i, \kappa_{\text{top}_i}, \kappa_{f_i} = \kappa_{\text{top}_i}, \Delta_i)$, and we attempt to connect $\hat{G}_1^{\text{cut}}, i = 1, 3$ with a

third G_3 curve. If no such third curve exists, we slowly decrease the value of $\sigma_{\max}$, and go back to Step 1. Figure 4.4 (right) illustrates an example where $\sigma_{\max}$ is decreased until $\hat{G}_i^{\text{cut}}, i = 1, 3$ can be connected with a third curve. Minimum arc-length is preserved by finding the largest feasible value of $\sigma_{\max}$ that allows for such a connection.

The second problem arises if no Dubins' solution can be found in Step 4. Similar to overlap, this arises if the centers $\boldsymbol{x}_{c_i}$ are too close together. For this reason, we again decrease the value of $\sigma_{\max}$ and return to Step 1. At the end of Steps 1-7, the process described above will have computed a G^3 curve $P_{\rho_{\max}}(s)$ that solves (4.4).

Remark 4.4.4. Critically, observe that to store a solution path between start and goal states computed using the above methods it is sufficient to store only the values $\rho_{\max}$, the top curvature κ_{top_i} and the value Δ_i for each of the potentially three curves, and the final curvatures of the first and last curves (i.e., 0 if $\hat{G}_1, \hat{G}_2$ can be connected with a straight line, and κ_{top_i} otherwise). Indeed, from (4.1), (4.11), (4.12) the resulting path can easily be obtained from these 9 values.

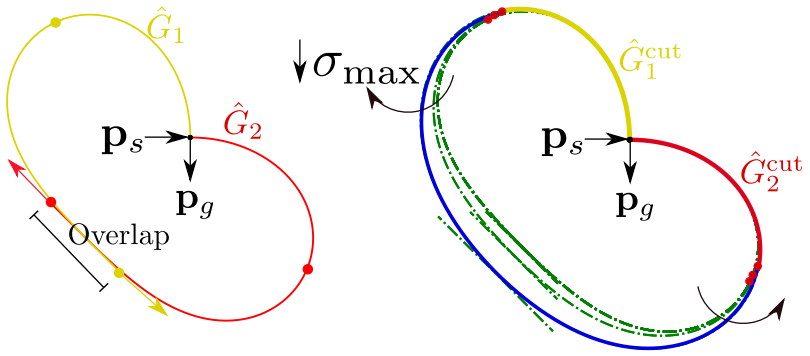


Figure 4.4: Step 7. (Left) Illustration of overlap. start and goal configurations too close together resulting in curve $P2$ terminating behind $P1$. (Right) Stretching of curves $P1, P2$ by decreasing $\sigma_{\max}$ until $P1, P2$ can be connected by third curve.

4.5 Computing Velocity Profiles

In this section we describe how to compute a longitudinal jerk profile $\beta(s)$ that minimizes the cost function in (4.10) over a fixed G^3 path $\pi_\rho(s)$. Similar to the procedure outlined

in [11], we approximate the continuous optimization problem by a discrete problem that asymptotically approaches the continuous version.

As discussed above, a velocity profile on a fixed path is equivalent to a re-parametrization of the path from arc-length to travel time. As such, a velocity profile along a G^3 path must be at least twice differentially continuous to produce a C^3 trajectory. For this reason, we offer a method by which a continuous piece-wise cubic velocity profile may be computed.

Given a fixed path $\pi_\rho(s)$ with final arc-length s_f, the high-level idea is to select $N+1$ arc-lengths s_i from $s_0 = 0$ to $s_N = s_f$ where $N \geq 4$, and then to select $N+1$ values $\beta(s_i)$. The value of $\beta(s)$ between sequentially sampled arc-lengths s_i, s_{i+1} is then given by the linear function passing through $\beta(s_i), \beta(s_{i+1})$.

Let $\boldsymbol{s} = [s_0, \ldots, s_N]$ denote $N+1$ arc-lengths with $s_0 = 0, s_N = s_f$. Further, let $\boldsymbol{B} = [b_0, b_1, \ldots, b_N]$ denote a vector of $N+1$ real numbers. This vector will serve as values of $\beta(s)$ at fixed arc-lengths $s_0, s_1, \ldots, s_N$. Given any vector $\boldsymbol{B}$, we may compute a velocity profile, $v^{\boldsymbol{B}}(s)$, longitudinal acceleration profile $\alpha^{\boldsymbol{B}}(s)$, and longitudinal jerk profile $\beta^{\boldsymbol{B}}(s)$ recursively as follows:

$$\begin{aligned} v^{\boldsymbol{B}}(s) =& v^{\boldsymbol{B}}(s_i) + \alpha^{\boldsymbol{B}}(s_i)(s - s_i) \\ &+ \frac{b_i(s-s_i)^2}{2} + \left(\frac{b_{i+1} - b_i}{s_{i+1} - s_i}\right)\frac{(s-s_i)^3}{6}, \\ \alpha^{\boldsymbol{B}}(s) =& \alpha^{\boldsymbol{B}}(s_i) + b_i(s - s_i) + \left(\frac{b_{i+1} - b_i}{s_{i+1} - s_i}\right)\frac{(s-s_i)^2}{2}, \\ \beta^{\boldsymbol{B}}(s) =& b_i + \left(\frac{b_{i+1} - b_i}{s_{i+1} - s_i}\right)(s - s_i), \end{aligned} \tag{4.19}$$

for all $s \in [s_i, s_{i+1}]$ and for all $i = 0, \ldots, N-1$. Observe that no bounds are placed on $d\beta^{\boldsymbol{B}}(s)/ds$. Therefore, for sufficiently small values of $s_{i+1} - s_i, i = 0, \ldots, N-1$ and large values of $d\beta^{\boldsymbol{B}}(s)/ds$, the equations in (4.19) can be used to model continuous piece-wise quadratic velocity profiles with piece-wise constant $\beta(s)$. Using the notation

$$\begin{aligned} S_1(i) &= 1/6(s_{i-1} - s_i)^2, \\ S_2(i) &= 1/6(s_{i-2} - s_i)(s_{i-2} + s_{i-1} - 2s_i), \\ S_3(j,i) &= 1/6(s_{j-1} - s_{j+1})(s_j + s_{j-1} - 3s_i + s_{j+1}), \\ S_4(i) &= 1/2(s_i - s_{i-1}), \\ S_5(i) &= 1/2(s_{i+1} - s_{i-1}), \end{aligned}$$

we compute $v^{\boldsymbol{B}}, \alpha^{\boldsymbol{B}}, \beta^{\boldsymbol{B}}$ at each value s_i as linear combinations of the vector $\boldsymbol{B}$:

$$\begin{aligned} v^{\boldsymbol{B}}(s_i) &= v_s + b_i S_1(i) + b_{i-1} S_2(i) + \sum_{j=1}^{i-2} b_j S_3(j,i), \\ \alpha^{\boldsymbol{B}}(s_i) &= b_i S_4(i) + \sum_{j=1}^{i-1} b_j S_5(j), \;\; \beta^{\boldsymbol{B}}(s_i) = b_i. \end{aligned} \tag{4.20}$$

We set $b_0 = 0, b_N = 0$. This is to ensure that when two trajectories are concatenated by a local planner, the resulting β-profile is continuous. Finally, we compute b_{N-2}, b_{N-1} in terms of $b_i, i = 1, \ldots, N-3$ to ensure that $\alpha(s_f) = 0, v(s_f) = v_f$. This is done by solving the linear system of equations $v^{\boldsymbol{B}}(s_N) = v_f, \alpha^{\boldsymbol{B}}(s_N) = 0$, for b_{N-1}, b_{N-2} where $v^{\boldsymbol{B}}(s_N), \alpha^{\boldsymbol{B}}(s_N)$ are given by (4.20). Equation (4.20) also facilitates the development of linear inequalities for the vector $\boldsymbol{B}$ to encode the constraints $0 \leq v^{\boldsymbol{B}}(s_i) \leq v_{\max}, |\alpha^{\boldsymbol{B}}(s_i)| \leq \alpha_{\max}, |\beta^{\boldsymbol{B}}(s_i)| \leq \beta_{\max}$.

Let $\boldsymbol{B}' = [b_1, \ldots, b_{N-3}]$ denote a decision vector of way-point values of longitudinal jerk. From $\boldsymbol{B}'$, we compute $\boldsymbol{B} = [b_0, b_1, \ldots, b_N]$ by setting $b_0 = b_N = 0$, and computing b_{N-1}, b_{N-2} by solving a system of linear equations to ensure the boundary constraints of the velocity profile. To select a decision vector, we solve the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{B}' \in [-\beta_{\max}, \beta_{\max}]^{N-3}} & C(\pi_\rho(s), v_{\beta^{\boldsymbol{B}}}(s)) \\ s.t., & \forall s_i \in \boldsymbol{s}, \\ 0 \leq & v^{\boldsymbol{B}}(s_i) \leq v_{\max}, \\ & |\alpha^{\boldsymbol{B}}(s_i)| \leq \alpha_{\max}, \\ & |\beta^{\boldsymbol{B}}(s_i)| \leq \beta_{\max}. \end{aligned} \tag{4.21}$$

For a fixed path $\pi_\rho(s)$, the integrand of $C(\pi_\rho(s), v_{\beta^{\boldsymbol{B}}}(s))$ in (4.8) reduces to a non-convex polynomial expression of $v^{\boldsymbol{B}}(s), \alpha^{\boldsymbol{B}}(s), \beta^{\boldsymbol{B}}(s)$. Thus, together with the explicit definitions of $v^{\boldsymbol{B}}(s), \alpha^{\boldsymbol{B}}(s), \beta^{\boldsymbol{B}}(s)$ in terms of $\boldsymbol{B}$ in (4.19), (4.20), first and second order derivatives of the cost with respect to $\boldsymbol{B}'$ can be easily computed (though these expressions are omitted for brevity). The optimization problem in (4.21) is an instance of non-convex, non-linear optimization with linear constraints, and can be solved quickly in practice using solvers like IPOPT [110] or SNOPT [33].

Remark 4.5.1. Paths computed using our methods require only 9 constants to describe (see Remark 4.4.4). Moreover, the velocity profiles here require only $N + 1 \geq 5$ additional constants. We typically use $N = 10$ (which works well in practice) for a total of 19 constants to describe a trajectory. This is typically far less than what is required to

describe numerically derived paths in the absence of analytic expressions. For example, a 10 meter trajectory with (x, y, θ, v) samples spaced every 0.1 meters would require 20 times more space to store than the proposed method.

4.6 Evaluation

We now demonstrate the benefits of our approach. We begin by illustrating how the techniques developed here can be used to anticipate the driving styles of several archetypal users, namely, a comfort-favoring (comfort) user who prefers low speed and high comfort, a moderate user who favors a mix of speed and comfort, and a speed-favoring (speed) user. Next, we compare the theoretical cost of paths generated using the proposed method to those from [6]. As implied by Theorem 4.3.1, any path that solves (4.9) is a G^3 path. It is for this reason that we compare our methods for generating trajectories to those proposed in [6], the current state-of-the-art G^3 path generation technique.

Finally, we measure tracking error and final cost of our proposed trajectories using Matlab's seven degree-of-freedom simulator with default road friction and input noise, and built-in Stanley controller [47]. Our methods were encoded entirely in Python 3.7 (Spyder), and IPOPT was used in Python to solve the OP in (4.21) to calculate velocity. The results were obtained using a desktop equipped with an AMD Ryzen 3 2200G processor and 8GB of RAM running Windows 10 OS.

4.6.1 Setup

Unit-less Weighting The features in the cost function (4.8) represent different physical quantities. In order for our cost function to represent a meaningful trade-off between these features for given weights, a scaling factor is included in the weight [48]. Similar to the scaling techniques in [38], we let $\hat{C}_m, m \in \{a, t, y, \mathcal{J}\}$ denote the cost of the trajectory that solves (4.9) given weight $w_m = 1$, $w_n = 0$, $\forall n \in \{a, t, y, \mathcal{J}\}$, $n \neq m$. We then scale the weights as

$$\hat{w}_m = \frac{w_m}{\hat{C}_m} \sum_{m \in \{a, \mathcal{J}, y, t\}} \hat{C}_m, \quad m \in \{a, \mathcal{J}, y, t\}.$$

This scaling process[1] has the following benefits: if all features are weighted equally, then the scaled cost of each feature $\hat{w}_m \hat{C}_m$ are equal. Also, it may be that for some $m \in \{a, \mathcal{J}, y, t\}$,

[1]NOTE: The scaling process described here is not included in the timing of our algorithm reported in Table 4.3

the feature cost C_m is very sensitive to changes in the trajectory in a neighborhood of the optimal trajectory. For example, IS jerk is the integral of a fifth order polynomial of v in (4.8) and is therefore highly sensitive to changes in v at high speeds, In these cases, if the scaling factor used any but the optimal value $\hat{C}_m$ for each feature, it would risk of over-reducing the weight on this feature.

Parameter Bounds For all experiments, the curvature parameter limits are given by [6]:

$$\kappa_{\max} = 0.1982m^{-1},\ \sigma_{\max} = 0.1868m^{-2}\ \rho_{\max} = 0.3905m^{-3}. \tag{4.22}$$

while the limits on velocity, acceleration, and instantaneous jerk are given, respectively, by [5]

$$v_{\max} = 100km/hr,\ a_{\max} = 0.9m/s^2,\ b_{\max} = 0.6m/s^3. \tag{4.23}$$

4.6.2 Evaluation

Qualitative Analysis To qualitatively evaluate our methods, we analyse the trajectories for three archetypal users: a speed user, who prefers low travel time over comfort, an intermediate user who values comfort and speed equally, and a comfort user who emphasizes comfort over speed. To simplify the comparison of these users, we combine the scaled but unweighted features IS acceleration, IS jerk, and IS yaw into a *discomfort cost*, and compare this to *time cost* for each user:

$$\text{discomfort cost} = \sum_{m\in\{a,\mathcal{J},y\}} \hat{w}_m\hat{C}_m,\ \text{time cost} = \hat{w}_m\hat{C}_t.$$

Given the simplistic nature of the speed, intermediate, and comfort archetypes, the expected form of their favored trajectories is apparent: the speed user wishes to minimize travel time at the expense of comfort. Therefore, she will favor trajectories featuring short paths and high velocities. The comfort user, on the other hand, prefers comfortable trajectories over short travel times. Finally, the intermediate driver should favor a balance of the two other users. In this experiment, the start-goal configurations for each user are:

$$p_s = [0, 0, 0, 0],\ p_f = [30, 105, 0, 0.1695],$$

with $v_s = 12m/s, v_g = 13m/s$. The weights for each user are:

$$\begin{aligned} \text{speed User } & w_t = 0.9, w_y = w_{\mathcal{J}} = w_a = 0.033, \\ \text{Intermediate User } & w_t = w_y = w_{\mathcal{J}} = w_a = 0.25 \\ \text{comfort User } & w_t = 0.01, w_y = w_{\mathcal{J}} = w_a = 0.33. \end{aligned} \tag{4.24}$$

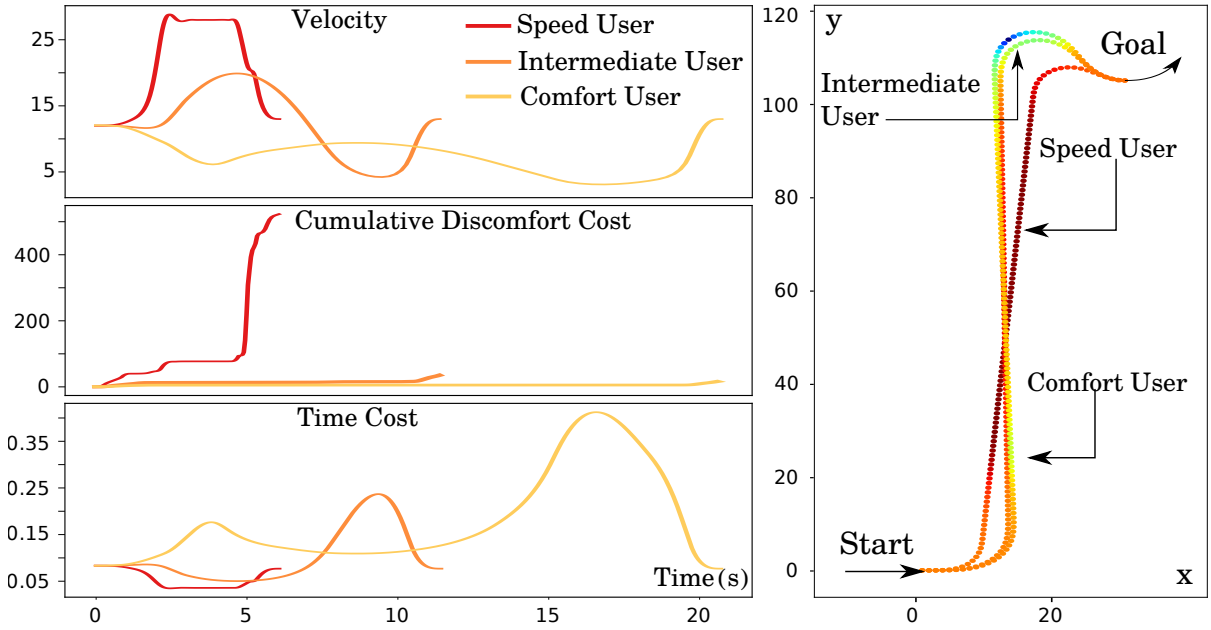


Figure 4.5: Comparison of three users. Left: velocity (m/s) (top), time cost (mid) and discomfort cost (bottom) for three users. Right: trajectories.

Thus, the speed user places a high weight on time, and a small weight on IS yaw rate, IS Jerk, and IS acceleration (i.e., small weight on discomfort), while the comfort user places a low weight on time, and the intermediate user places an equal weight on all features. The results are illustrated in Figure 4.5. The left image illustrates the velocity profile (top), discomfort cost (mid), and time cost (bottom) for the speed (red), intermediate (blue), and comfort (green) drivers. The right image illustrates the resulting trajectories for each user with color representing velocity. The shortest trajectory is that of the speed user. Observe that her path is short and velocities high, reflecting her preference towards short travel times. The longest trajectory is that of the comfort user, and we note that her trajectory features consistently low velocities and a long path. This tendency towards longer path length, allows the comfort user to change velocities over a longer time period allowing her to reach lower velocities with lower acceleration and jerk. This is reflected in the low discomfort cost of the comfort user. The middle trajectory is that of the intermediate user, who decreases her velocity on the turn to minimize jerk/acceleration, but then speeds up again on the straight portion of the path.

A further example of the three archetypes is given in Figure 4.6 in which a lane change maneuver is preformed from $(0, 0, 0, 0)$ to $(50, 6, 0, 0)$ with an initial and final velocity of

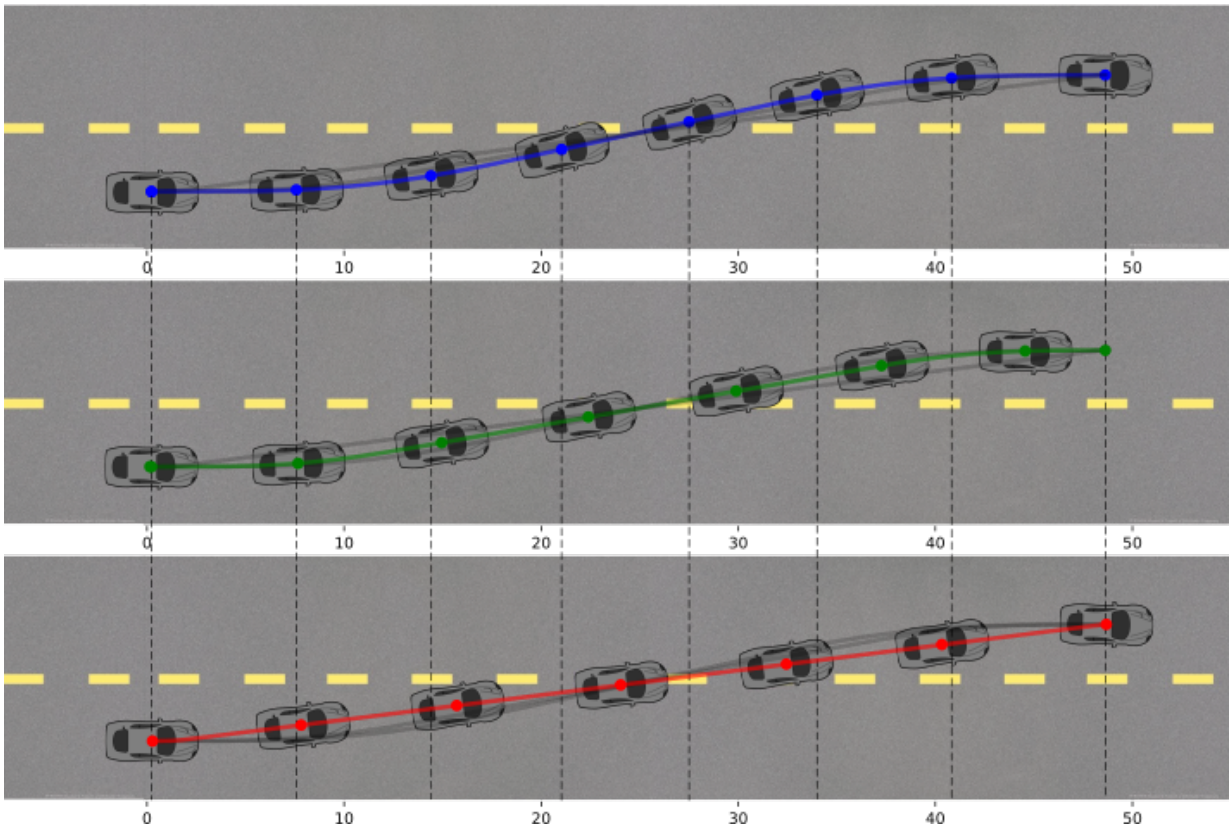


Figure 4.6: Example reference trajectories for a lane change maneuver given three minimization objectives: discomfort (top), time (bottom), and a mix between time and discomfort (mid). Initial and final speeds are fixed at 10 m/s, cars are drawn ever 0.75 seconds.

10 m/s. Cars are drawn ever 0.75 seconds, with the dotted vertical lines representing horizontal position of the top-most car at each time step. The comfort driver has an average speed of 9.4 m/s and completes the maneuver in 5.30 seconds, while the mixed and speed drivers have average speeds and final times of (10.09 m/s, 4.91 seconds), and (10.80 m/s, 4.55 seconds), respectively. Observe that the car following the red trajectory arrives at the goal almost a full time-step ahead of the blue-trajectory car.

Path Evaluation We now evaluate the quality of the paths generated using our technique. To this end, we isolate the effect of the path on the final cost by comparing two methods for generating trajectories:

- *Ours*: Paths and velocity profiles are computed using the methods detailed in this chapter.
- *Benchmark*: velocity profiles are computed using the techniques outlined here, but paths are computed from [6].

We define the % Savings as the difference in cost (given in (4.8)) for trajectories generated using Benchmark and Ours as a percentage of the cost of Benchmark-based trajectories.

		Avg. Percent Savings (%)
Dominant Feature	IS Time	20.00 (20.7)
	IS Yaw	26.24 (24.8)
	IS Acceleration	30.26 (24.2)
	IS Jerk	64.09 (26.7)
	Blended	42.06 (25.0)
Initial/Final Velocity	Low $(0, v_{\max}/3]$	39.41 (29.9)
	Med $(v_{\max}/3, 2v_{\max}/3]$	38.57 (29.0)
	High $(2v_{\max}/3, v_{\max}]$	33.50 (27.8)

Table 4.1: Average cost savings. Breakdown by dominant feature and initial/final velocity.

For this experiment, 1300 start-goal pairs were randomly generated with initial and final curvatures ranging from $-\kappa_{\max}$ to $\kappa_{\max}$, initial and final velocities (taken to be equal) ranging from 0 to $v_{\max}$, initial and final x, y-values ranging from $-20m$ to $20m$, and weights ranging from 0 to 1. Initial and final velocities were chosen to reflect the Euclidean distance between start-goal pairs, and pairs with no feasible solution were disregarded. The results of these experiments are tabulated in Table 4.1. Here, a feature (IS time, IS yaw, IS acceleration, IS jerk) is said to be *dominant* if the corresponding weight is greater than 0.5. If no weight is greater than 0.5, the features are said to be *blended*. It was found that the average percent savings was 36.35% across all experiments. Moreover, if the velocity is not optimized using our method, but is assumed to be constant instead, we see see a 43.3% savings. As Table 4.1 implies, the largest savings is typically found when IS jerk is the dominant feature. This implies that the techniques presented here have greater value for users who prefer comfort to low travel times, but can still reduce costs by 20% on average for such users. A detailed view of the distribution of the percent savings for the experiments is given in Figure 4.7. Observe that the instances where the percent savings are substantial ($> 50\%$) are not infrequent. In particular, in 34% of experiments, the

savings was at least 50%, while in 17% of experiments, the savings was at least 70%, and 10% of experiments had a savings of at least 80%.

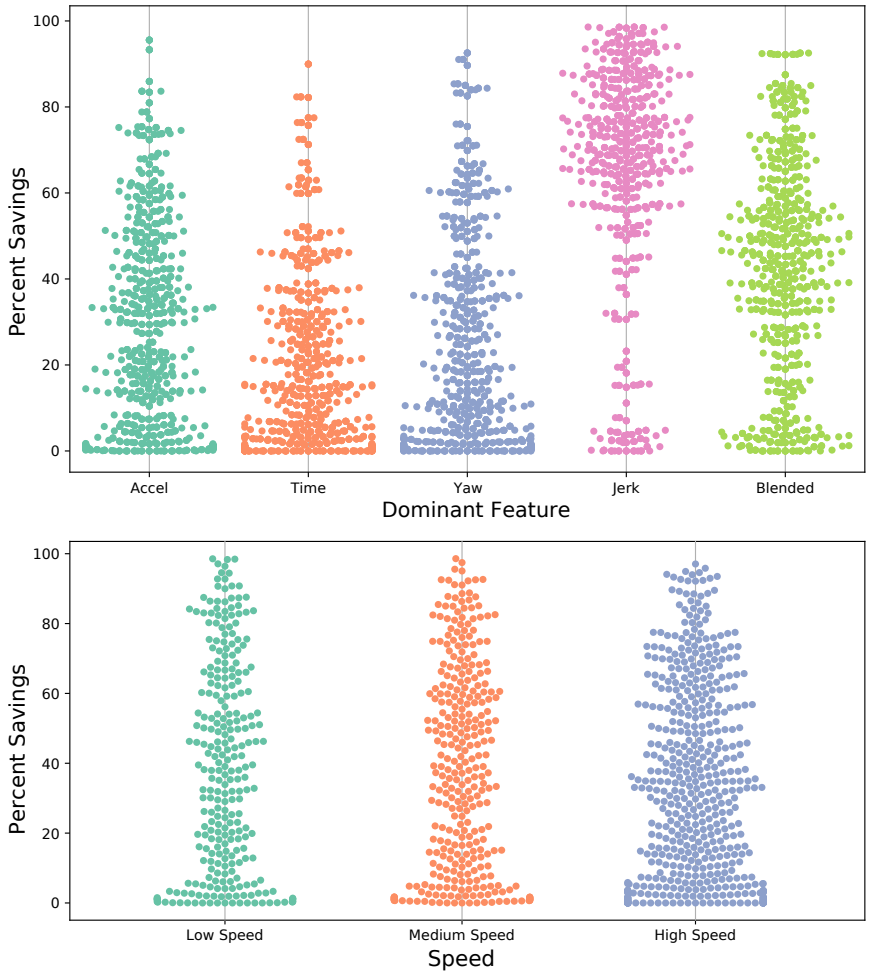


Figure 4.7: Percent savings distribution by dominant feature (top), and initial/final velocity (bottom).

Tracking Error and Simulation Cost The above results imply that the methods proposed herein can be used to compute reference trajectories that result in substantial cost reduction if tracked perfectly by a vehicle. However, it is highly unlikely that any reference will be perfectly tracked. While we do not propose a tracking controller in this chapter (nor does this chapter propose a local planner), we will address two important questions. First

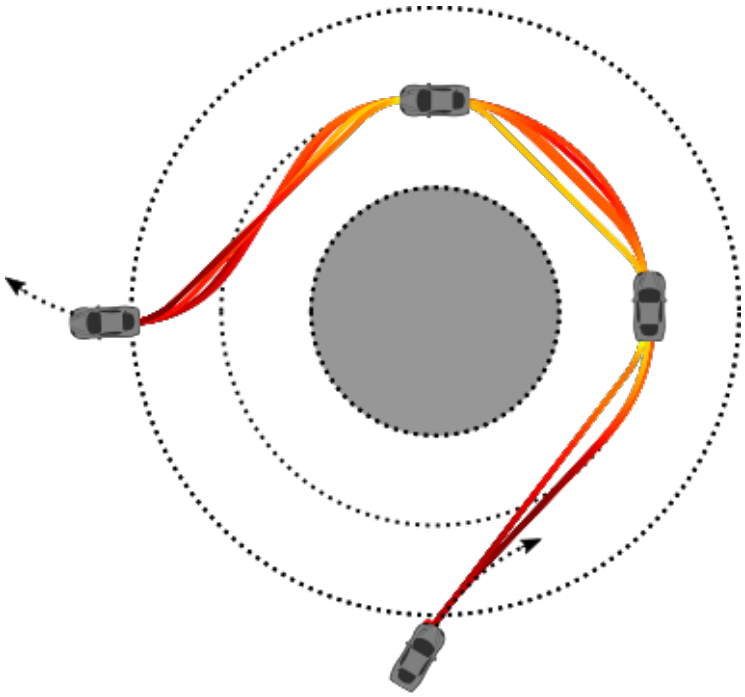


Figure 4.8: Optimal trajectories for weights between way-points of a roundabout. Cars represent fixed way-points (position, curvature, velocity) while color gradient of each trajectory represents velocity.

will the imperfections in the vehicles controller negate the theoretical cost savings? Second, even if there is still a reduction in cost using our method, does this come at the expense of the tracking error? That is, are reference trajectories generated using our method harder to track than the Benchmark method? In this section, we use a basic out-of-the-box feedback controller that has not been optimized, and we do not re-plan reference trajectories. This is to illustrate that even with a sub-optimal controller, there is still substantial cost savings. Moreover, there is typically even a *reduction* in the tracking error.

In this section. We again consider the methods "Ours" and "Benchmark" defined above. The experiments in this section were carried out using MATLAB's seven degree of freedom driving simulator in its autonomous driving toolbox. All car dimensions, force coefficients, and noise parameters were left at their default values. The reference trajectories were tracked using MATLAB's Stanley controller with default gains. Here, the maximum error is the maximum lateral error, and the average error is the integral of the lateral error normalized to arc-length of the path. We consider three maneuvers between start goal pairs

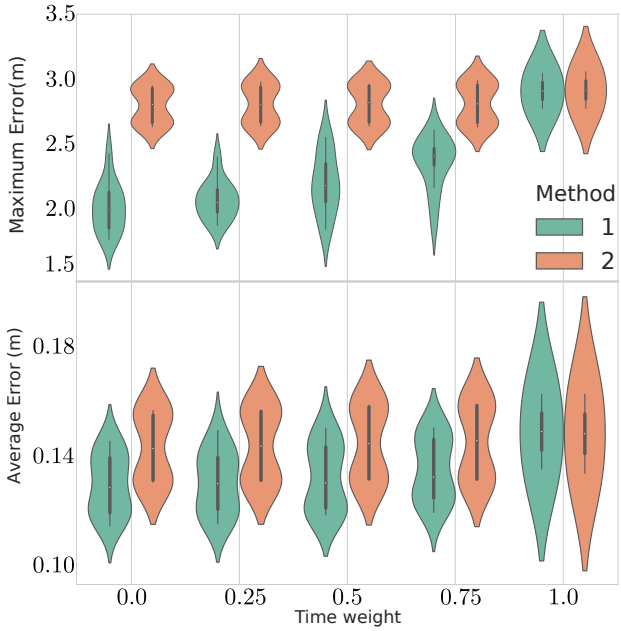


Figure 4.9: Average maximum errors (top) and average average errors (bottom) for Methods 1 (Ours) and 2 (Benchmark) categorized by time weight.

chosen to coincide with possible way-points of a roundabout. Dimensions and velocities for this problem are in accordance with U.S. department of transportation guidelines [96, Chapter 6]. The single-lane roundabout, illustrated in Figure 4.8, has lane radius (middle circle) of 45 ft, The initial way-point for this experiment is on a connecting road 65 ft away from the center of the roundabout. We assume an initial velocity of 17 mph. At the second and third way-points, the velocity is assumed to be 14 mph which increases to 17 mph at the final way-point. Trajectories between these way-points were planned for 50 weights: 5 values of the time weight w_t, (0, 0.25, 0.5, 0.75, 1) and 10 randomly selected values of the other weights for each value of w_t (ensuring that the sum of the weights equaled 1). Trajectories and their lateral errors for one example weight ($w_{\mathcal{J}} = 1, w_t = w_a = w_y = 0$) is shown in Figure 4.10. The left-most images illustrate the trajectories obtained using the two methods. Blue, red, and green paths represent theoretical, tracked, and driven paths, respectively. Short line segments between red and green paths connect driven configurations with the reference configurations being tracked at each time step. Observe that the high peaks in error (right-most images) experienced by cars following a Benchmark reference trajectory, are spread more evenly along the trajectory for cars tracking Ours trajectories. This is because paths generated using Ours will have more

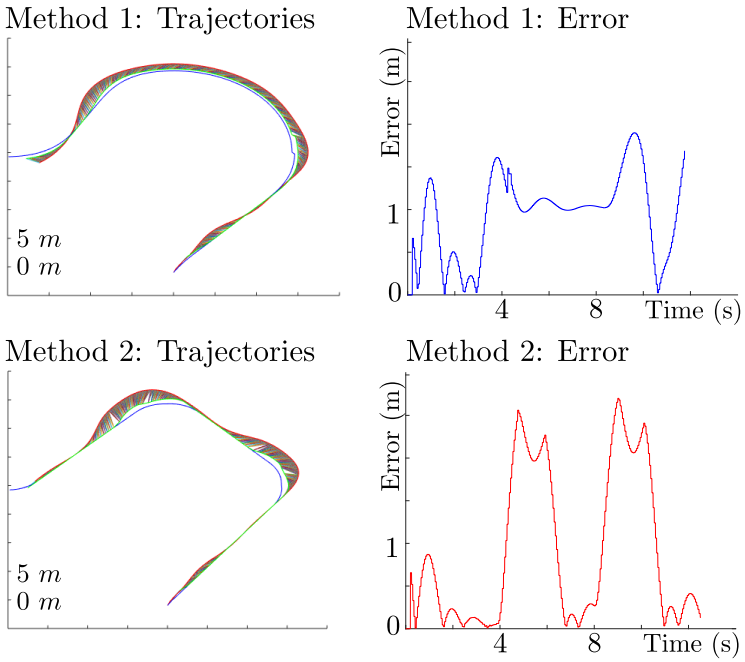


Figure 4.10: (Left) Simulated trajectories for example roundabout maneuver. Blue, green, and red lines are reference, tracked, and driven paths. Connecting lines show tracking error. (Right) Lateral Error over time. Methods 1 (Ours) and 2 (Benchmark).

gradual changes in curvature given the example weight. For each value of w_t, the values of maximum error and average error of each experiment are tabulated in Figure 4.9, and Table 4.2. Observe that (as implied by the example in Figure 4.10), savings in every metric is highest for weights that favor comfort over time. Observe further, that the maximum lateral tracking error can be reduced by approximately 0.8m on average for certain weights. The roundabout radius and velocities were scaled upwards (again following the guidelines of [96]). However, the results were very similar to those reported above.

Time weight (w_t)	Cost Savings (%)	Max Err (m)		Avg Err (m)	
		O	B	O	B
0	35.8	2.05	2.82	0.13	0.14
0.25	18.7	2.11	2.83	0.13	0.14
0.5	13.7	2.33	2.85	0.13	0.14
0.75	8.9	2.40	2.85	0.14	0.14
1	1.0	2.75	2.77	0.15	0.15

Table 4.2: Average cost savings, maximum lateral error and average lateral error for two methods (O = Ours, B = Benchmark), categorized by time weight.

Run-time The focus of this work was on improving the quality of computed trajectories given user weights. As such, we did not focus on optimizing for run time. However, we believe that with proper software engineering, one could substantially reduce the run-time of our procedure though it may still exceed the state-of-the-art (SOTA). Run-times for

Sub-routine	SOTA source	Run-times (s)	
		SOTA (C)	Ours (Python)
Calculate a path (Step 1)	[6]	6.8×10^{-5}	4.3×10^{-3}
Optimize velocity (Step 2)	[46]	2.2×10^{-4}	1.2×10^{-2}
Total (recursive Step 1 & 2)	[6] & [46]	1.2×10^{-3}	8.5×10^{-2}

Table 4.3: Comparison of run-times. Our methods (Ours) compared against state-of-the-art (SOTA) for each sub-routine of the procedure.

each of the two steps in our procedure are compared with those of the SOTA in Table 4.3 which also includes the total time. Reference [46] as the SOTA for velocity profile planning given a path due to its use of third order trapezoidal methods. Trapezoidal methods (and variants thereof) are a widely used method for velocity profile planning [28, 92]. Variants of this method use higher-order velocity models than those employed by [46], improving the quality at the expense of computation time. We therefore use computation times reported in [46] as a lower bound on trapezoidal velocity method variants. We observe that the SOTA is on the order of 100 times faster than the methods proposed here. However, our algorithm is encoded entirely in Python which tends to run 10-100 times slower than C for iterative procedures like ours.

4.7 Discussion

We consider the problem of computing trajectories between start and goal states that optimize a trade off between comfort at time. We offer a simplified approximation of optimization problem (4.9) that replaces the two continuous functions with $N+2$ constants. Methods to compute paths given one of these constants, and a velocity profile given this path and the $N+1$ remaining constants are also provided. The resulting technique is verified with numerical examples.

Chapter 5

High-Dimensional Lattice Planning with Optimal Motion Primitives

Research in the chapter was published in the proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS) 2019 [12].

5.1 Introduction

In this chapter, we address the Minimum $t-$Spanning Control Set (MTSCS) problem 2.3.8 and incorporate a MTSCS in a local planner. For clarity, the contributions of this chapter are repeated here from Section 1.1.4.

1. We prove that the decision version of the MTSCS problem 2.3.9 is NP-hard.
2. We propose a mixed integer linear programming (MILP) formulation of the MTSCS problem 2.3.8. This represents the first known non-brute force approach to solving the MTSCS problem. The result is a control set of minimal size that generates motions that are continuous in all states and have bounded $t-$factor sub-optimality. Though this formulation does not scale for large lattices, we observe that control sets need only be computed once once, offline.
3. We present an A*-based algorithm to compute feasible motions for difficult maneuvers in both parking lot and highway settings. The algorithm is an amalgamation of two other established algorithms: Bi-Directional A* [14, 114], and weighted A* [91] but can accommodates off-lattice start and goal configurations up to a specified tolerance.

4. We provide a novel algorithm that eliminates redundant vertices along motions computed using a lattice. This algorithm is based on shortest paths in directed acyclic graphs and eliminates excessive oscillations, a common critique of lattice-based motion planning [76]. The algorithm runs in time quadratic in the number of motion primitives along the input motion.

5.2 Main Results

In this section we present a MILP formulation of Problem 2.3.8, and algorithms to compute and smooth motions using the control sets returned by Problem 2.3.8. This section references the notation established in Section 2.3. We begin with a motivating theorem.

Theorem 5.2.1 (NP-Hardness of the MTSCS Problem 2.3.9)**.** *The decision version of the Minimum $t-$Spanning Control Set problem in 2.3.9 is NP-hard.*

Proof. We prove this result by offering a reduction from the Vertex Cover Problem in 2.1.1. Given an instance of the Vertex Cover problem, that is, a graph $G' = (V', E')$ with $V' = \{v'_1, \ldots, v'_n\}, E' = \{e'_1, \ldots, e'_m\}$, and a natural number K, we construct an instance of Problem 2.3.9 by constructing the tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W})$ and real number $t \geq 1$. We will show that G' has a vertex cover of size no more than K if and only if L has a MTSCS of size no more than $K + 2|E'|$.

We begin by defining the configuration space $\mathcal{X}$. Let $\mathcal{X}$ be the set of configurations of the form $(x, y) \in \mathbb{R}^2$ denoting position. Next, let $\mathcal{O} = \{o = (0, 0)\}$ be the starting set with a single start. For any configurations i, j we let $i \cdot j$ be the vector addition of i and j. That is, if $i = (i^x, i^y), j = (j^x, j^y)$, then $i \cdot j = (i^x + j^x, i^y + j^y)$. We construct a lattice L for the configuration space in time polynomial in $|E'| + |V'|$ as follows:

1. For each graph vertex $v'_i \in V$, define a lattice vertex:

$$v_i = (v_i^x, y_i^y) = \left(\cos\left(\frac{(i-1)\pi}{3m}\right), \sin\left(\frac{(i-1)\pi}{3m}\right)\right).$$

This is illustrated in Figure 5.1 (b) for the input graph in Figure 5.1 (a).

2. For each graph edge (v'_i, v'_j) let $v_i = (v_i^x, v_i^y), v_j = (v_j^x, v_j^y)$ be the lattice vertices associated with v'_i, v'_j, respectively. We define six lattice vertices:

$$v_{i-j} = (v_i^x - v_j^x, v_i^y - v_j^y)$$
$$v_{j-i} = (v_j^x - v_i^x, v_j^y - v_i^y)$$
$$v_{i+j} = v_i \cdot v_j = (v_i^x + v_j^x, v_i^y + v_j^y).$$

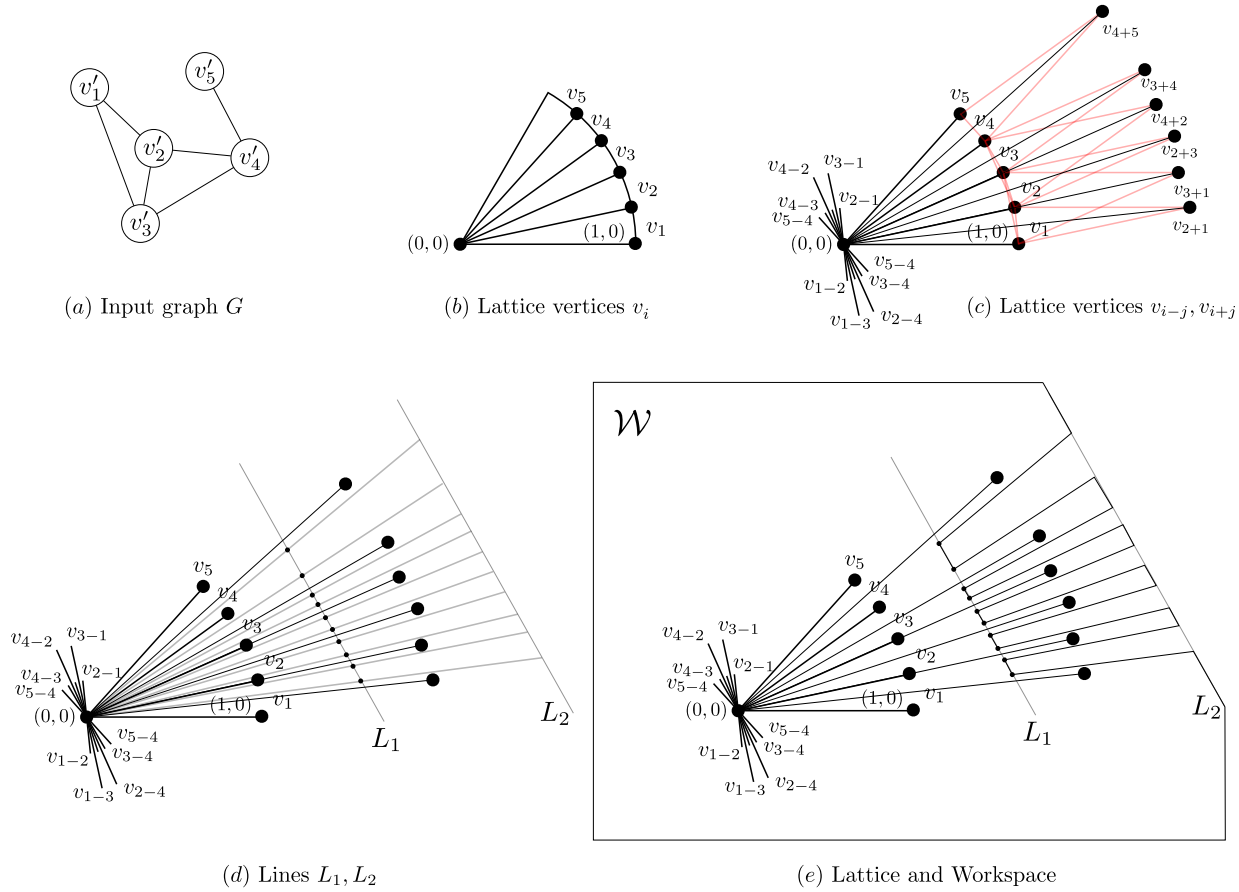
This step is illustrated in Figure 5.1 (c).

Observe that each vertex is distinct. Indeed, vertices of the form v_i lie in the first quadrant along the unit circle, while vertices of the form v_{i+j} lie in the first quadrant and have minimum length $\sqrt{3} > 1$. Further, vertices of the form v_{i-j}, v_{j-i} lie in the third or fourth quadrant and are all distinct since no two non-diametral secant lies of an upper-half circle are equal. Next, we define the cost of each motion. Let c_m denote the cost of the motion from o to the lattice vertex m where m can take the form $i, i-j, i+j$. Further, let

$$c_m = \begin{cases} 1 \text{ if } m \text{ is of the form } i, \\ 0.1 \text{ if } m \text{ is of the form } i-j \\ 1.91 \text{ if } m \text{ is of the form } i+j. \end{cases}$$

Finally, we define a workspace $\mathcal{W}$ such that motions between vertices of the form v_{i+j} will exit the workspace. This can be done in time polynomial in $|E| + |V|$ as follows: define a line segment L_1 connecting configurations $(1.5, 0)$ and $(1.5\cos(\pi/3), 1.5\sin(\pi/3))$. Define another line segment L_2 connecting configurations $(2.5, 0)$ and $(2.5\cos(\pi/3), 2.5\sin(\pi/3))$. These lines are illustrated in Figure 5.1 (d). Next, determine where the lines connecting the origin and vertices of the form v_{i+j} intersect the line segment L_1. This splits L_1 into sub-segments $L_1^i, i = 1, \ldots, |E'|$. In each segment L_1^i, determine two points on the line, one 1/4 of the way along L_1^i, and the other 3/4 of the way along. Connect each point to the origin via a straight line and determine where this straight line intersects L_2. These steps are illustrated in Figure 5.1 (d). Finally, connect L_1 and L_2 is a saw-tooth manner by connecting the determined points in L_1 with their corresponding points in L_2 as in Figure 5.1 (e). Since the minimum distance of vertices of the form v_{i+j} from the origin is $\sqrt{3}$, line segments connecting two such vertices will necessarily pass through the boundary of $\mathcal{W}$. Further, since vertices of the form v_i lie a distance of 1 from the origin, line segments connecting two such vertices will necessarily lie entirely in $\mathcal{W}$. Finally, let $t = 1.1$.

Let p_m denote the motion from the origin to v_m for any $v_m \in L$. Then $c(p_m) = c_m$. Observe that if E is a MTSCS of the lattice above, then E must contain every motion of



(*a*) Input graph G (*b*) Lattice vertices v_i (*c*) Lattice vertices v_{i-j}, v_{i+j}

(*d*) Lines L_1, L_2 (*e*) Lattice and Workspace

Figure 5.1: Constructing a Lattice and Workspace.

the form p_{i-j}. Indeed, if $p_{i-j} \notin E$ for some vertex of the form v_{i-j}, then $d^E(v_{i-j})/c_{i-j} \geq 0.2/0.1 = 2 > 1.1 = t$ since all motions have cost at least 0.1 and $c_{i-j} = 0.1$ for all motions of the form p_{i-j}. Observe further that if E contains motions of the form p_{i+j} from the origin to a vertex of the form v_{i+j}, then a new MTSCS (of equal size) can be constructed by replacing p_{i+j} with either p_i or p_j. Indeed, v_{i+j} cannot lie on a path using E to any other vertex v_{r+s} since the required motion would exit the workspace by construction. Moreover, v_{i+j} cannot lie on a path using E to any vertex of the form v_k or v_{r-s} since this would imply that $d^E(v_k)/c_k \geq (1.91 + 0.1)/1 = 2.1 > 1.1 = t$ and $d^E(v_{r-s}) \geq (1.91 + 0.1)/0.1 = 21 > 1.1 = t$ implying that E is not a MTSCS. Finally, if $E' = E - \{p_{i+j}\} \cup \{p_i\}$, then because $v_{i+j} = v_i \cdot v_j$, $d^{E'}(v_{i+j}) = (c_i + d^E(v_j))/c_{i+j} \leq (1 + 1 + 0.1)/1.91 = 2.1/1.91 < 1.1 = t$ implying that v_{i+j} is reachable within a factor of t using E'. Therefore, without loss of generality, we may assume that a MTSCS E contains all motions of the form p_{i-j} (of which

there are $2|E'|$) and no motions of the form p_{i+j} (since these can be replaced with motions of the form p_i without increasing the solution size).

Finally, observe that a set E containing all of the motions of the form p_{i-j} and no motions of the form p_{i+j} is a MTSCS of L if and only if for all $v_{i+j} \in L$, either $p_i \in E$ or $p_j \in E$. Indeed, suppose $p_i \in E$. Then because $p_{j-i} \in E$ and $v_i \cdot v_{j-i} = v_j$, we have $d^E(v_j)/c_j \leq (c_i + c_{j-i})/c_j = (1 + 0.1)/1 = 1.1 \leq t$ and v_j is reachable within a factor of t using E. Further, since $v_i \cdot p_j = v_{i+j}$, and v_j is reachable within a factor of t using E, $d^E(v_{i+j})/c_{i+j} \leq (c_i + d^E(v_j))/c_{i+j} \leq (1 + 1.1 \times 1)/1.91 < 1.1 = t$ and v_{i+j} is reachable within a factor of t using E. Therefore, if $p_i \in E$ and $v_{i+j} \in L$ then v_i, v_j, v_{i+j} are all reachable within a factor of t using E. If this holds for all $v_{i+j} \in L$, then L is $t-$spanned by E. On the other hand, if $v_{i+j} \in L$ and $v_i, v_j \notin E$ then E is not a MTSCS of the lattice. Indeed, if $v_i, v_j \notin E$, then because the cost of the minimal-cost motion is 0.1 and E contains no motions of the form p_{i+j}, it must hold that $d^E(v_{i+j})/c_{i+j} = (d^E(v_i) + d^E(v_j))/c_{i+j} \geq (1 + 0.1 + 1 + 0.1)/1.91 > 1.1 = t$ implying that v_{i+j} is not reachable within a factor of t using E, and E is not a MTSCS.

We have shown that there exists a MTSCS E of the lattice L of the following form: every motion of the form p_{i-j}, p_{j-i} (of which there are $2|E'|$ such motions) is in E, and where no motions of the form p_{i+j} are in E. Further, such a set is a MTSCS if and only if for each vertex of the form v_{i+j} of which there are $|E'|$, either $p_i \in E$ or $p_j \in E$. To complete the proof, let E be a MTSCS of the form just described, and let $\mathcal{V} \subset V'$ be given by $\mathcal{V} = \{v'_i \in V' : p_i \in E\}$. Then,

$$\begin{aligned} & E \text{ is a MTSCS of } L \text{ of the form described above} \\ \iff & \forall v_{i+j} \in L,\ p_i \in E \vee p_j \in E \\ \iff & \forall (v'_i, v'_j) \in E', v_i \in \mathcal{V} \vee v_j \in \mathcal{V} \\ \iff & \mathcal{V} \text{ is a vertex cover of } G. \end{aligned} \tag{5.1}$$

Noting that $|E| = |\mathcal{V}| + 2|E'|$ completes the proof. That is, there exists a vertex cover $\mathcal{V}$ of $G' = (V', E')$ with $|\mathcal{V}| \leq K$ if and only if there exists a MTSCS E of L with $|E| \leq K + 2|E'|$. □

5.2.1 MTSCS Problem: MILP Formulation

Theorem 5.2.1 is motivation to provide a MILP formulation of Problem 2.3.8. Let $(\mathcal{X}, L, \mathcal{O}, c)$ denote configuration space, lattice, start set, and cost of vertex-to-vertex motions in L, respectively. For any motion $p \in \mathcal{B}$ (where $\mathcal{B}$ is given in (2.6)), let

$$S_p = \{(i, j) \ : \ i, j \in L,\ i \cdot p = j \text{ is a valid}\}.$$

Thus S_p is the set of all pairs $(i,j) \in L^2$ such that p the motion from i to j and $i \cdot q$ is valid. By definition of valid concatenations, there exists $o \in \mathcal{O}$ and $j' \in L$ such that p is the motion from o to j' implying that $(o,j') \in S_p$.

Let $E = \bigcup_{o \in \mathcal{O}} E_s$ be a solution to Problem 2.3.8. Let $G_{\text{Free}} = (L, \bar{E}, c)$ be the weighted directed graph with edges $\bar{E}$ given in (2.7). For each $o \in \mathcal{O}$ and each $(i,j) \in \bar{E}$, we make a copy $(i,j)^o$ of the edge. This allows us to treat edges differently depending on the starting vertex of the path to which they belong. For each $v \in L - \mathcal{O}$, a path using E from o to v, $\pi^E(o,v)$, may be expressed as a sequence of edges $(i,j)^o$ where $(i,j) \in \bar{E}$. For each $o \in \mathcal{O}$ we construct a new graph

$$T^o = (L - \mathcal{O} \cup \{o\}, E_T^o), \tag{5.2}$$

whose edges E_T^o are defined as follows: let

$$\tilde{T}^o = \bigcup_{v \in L - \mathcal{O}} \pi^E(o,v).$$

Thus $\tilde{T}^o$ is the set of all minimal cost paths from o to vertices $v \in L - \mathcal{O}$ in the graph G_{Free}. These paths can be expressed as a sequence of edge copies $(i,j)^o$ where $(i,j) \in \bar{E}$. For each $v \in L - \mathcal{O}$, and for each $o \in \mathcal{O}$, if $\tilde{T}^o$ contains two paths π_1^E, π_2^E from o to v, determine the last common vertex j in paths π_1^E, π_2^E, and delete the copy of the edge in π_2^E whose endpoint is j from $\tilde{T}^o$. Let the remaining edges be the set E_T^o. The graph T^o in (5.2) is an Arborescence rooted at o (defined in Definition 2.1.2) which we will leverage in our MILP formulation. This is shown in the following Lemma:

Lemma 5.2.2 (Arborescence Lemma)**.** *Let $E = \bigcup_{o \in \mathcal{O}} E_s$ be a solution to Problem 2.3.8, and $G_{Free} = (L, \bar{E}, c)$ be the weighted directed graph with $\bar{E}$ given in (2.7). For each $o \in \mathcal{O}$, if T^o is given by (5.2), then T^o is an arborescence rooted at o. Further, $\forall v \in L - \mathcal{O}$, $d^E(o,v)$ is the length of the path in T^o to v.*

Proof. Let $o \in \mathcal{O}$. To show that T^o is an arborescence rooted at o, observe first that there is a path in T^o from o to all $v \in L - \mathcal{O}$. Indeed, if E solves Problem 2.3.8, then E $t-$spans L. In particular, there must be at least one path, $\pi^E(o,i)$ using E from o to each $v \in L - \mathcal{O}$ implying that $\pi^E(o,v) \in \tilde{T}^o$. Since E_T^o only deletes duplicate paths from $\tilde{T}^o$, there must still be a path in T^o from o to v. Furthermore, by construction of the edge set E_T^o, duplicate paths (with equal cost) are deleted thus ensuring the uniqueness of paths from o to each $v \in L - \mathcal{O}$ completing the proof that T^o is an arborescence rooted at o. Finally, the cost of the path in T^o from o to v is defined as the cost of the path $\pi^E(o,v)$ which is the distance using E from o to v by definition. □

Lemma 5.2.2 implies that E is a $t-$spanner of L if and only if $\forall o \in \mathcal{O}$ there is a corresponding arborescence T^o rooted at o whose vertices are $L - \mathcal{O} \cup \{o\}$, whose edges $(i,j)^o$ are copies of members of S_p for some $p \in E$, and such that the cost of the path from o to any vertex $v \in L - \mathcal{O}$ in T^o is no more than a factor of t from the optimal path from o to v. Indeed, the forward implication follows from Lemma 5.2.2, while the converse holds by definition of a $t-$spanner. From this, we develop four criteria that represent necessary and sufficient conditions for E to be a $t-$spanning control set of L:

Usable Edge Criteria: For any motion $p \in \mathcal{B}$ from a start $o \in \mathcal{O}$ to a vertex in $L - \mathcal{O}$, for any start $o' \in \mathcal{O}$, and for any $(i,j) \in S_p$, the copy $(i,j)^{o'}$ may belong to a path in $T^{o'}$ from o' to a vertex $i \in L$ if and only if $p \in E_o$. That is, a tree $T^{o'}$ may posses an edge $(i,j)^{o'}$ if and only if the motion p from i to j is in the control set (i.e., if $p \in E_o$ where $o = R(i)$, the relative start of i).

Cost Continuity Criteria: For any $o \in \mathcal{O}$, $p \in \mathcal{B}$, $(i,j) \in S_p$, and $(i,j)^o$ a copy of (i,j), if $(i,j)^o$ lies in the path in T^o to vertex j then $c(\pi^E(o,j)) = c(\pi^E(o,i)) + c(p)$. That is, the cost of the path from o to j in T^o is equal to the cost of the path from o to i plus the cost of the motion from i to j.

$t-$Spanning Criteria: The length of the path in T^o to any vertex $j \in L - \mathcal{O}$ can be no more than t times the length of the direct motion from o to j.

Arborescence Criteria: The set T^o must be an arborescence for all $o \in \mathcal{O}$.

We now present how to encode these four criteria. Let $|L| = n$ and all the vertices are enumerated as $1, 2, ..., n$ with $o \in \mathcal{O}$ taking the values $1, \ldots, m$ for $m \leq n$. For any control set $E \subseteq \mathcal{B}$ where $E = \bigcup_{o \in \mathcal{O}} E_o$, define $m(n-m-1)$ decision variables $y_p^o, p = m+1, ..., n$

$$y_p^o = \begin{cases} 1, & \text{if } p \in E_o \\ 0, & \text{otherwise.} \end{cases}$$

Thus y_p^0 is the decision variable indicating whether the motion p from o to some $j' \in L$ is in the set E_o. For each pair $(i,j) \in L^2$, and each $o \in \mathcal{O}$ let

$$x_{ij}^o = \begin{cases} 1 & \text{if } (i,j)^o \in T^o \\ 0 & \text{otherwise.} \end{cases}$$

That is, $x_{ij}^o = 1$ if $(i,j)^o$ (the copy of the edge (i,j) for start $o \in \mathcal{O}$) lies on a path from o to a vertex in the lattice. If a tuple (i,j) with motion p from i to j is such that $i \cdot p$ is not

valid – that is, if the motion p starting at i is not contained in the workspace, or if i is not generalized by $\mathcal{O}$, then we set $x_{ij}^o = 0$ for all $o \in \mathcal{O}$. Let z_i^o denote the length of the path in the tree T^o to vertex i for any $i \in L$, c_{ij} the cost of the direct motion from i to j for any $i, j \in L$, and let $L' = L - \mathcal{O}$. The criteria above can be encoded as the following MILP:

$$\begin{aligned}
&\min \ K && && (5.3a)\\
&s.t. \ \forall o \in \mathcal{O} && && (5.3b)\\
&y_p^o \leq K, && \forall p \in \mathcal{B} && (5.3c)\\
&x_{ij}^{o'} - y_p^o \leq 0, && \forall o' \in \mathcal{O}, \forall (i,j) \in S_p, \forall p \in \mathcal{B} && (5.3d)\\
&z_i^o + c_{ij} - z_j^o \leq M_{ij}^o (1 - x_{ij}^o), && \forall (i,j) \in L^2 && (5.3e)\\
&z_j^o \leq t c_{oj}, && \forall j \in L' && (5.3f)\\
&\sum_{i \in L} x_{ij}^o = 1, && \forall j \in L' && (5.3g)\\
&x_{ij}^o \in \{0,1\}, && \forall (i,j) \in L \times L' && (5.3h)\\
&y_p^o \in \{0,1\}, && \forall p \in \mathcal{B}, && (5.3i)
\end{aligned}$$

where $M_{ij}^o = t c_{oi} + c_{ij} - c_{oj}$. The objective function (5.3a) together with constraint (5.3c) minimizes $\max_{o \in \mathcal{O}} |E_s|$ as in Problem 2.3.8. The remainder of the constraints encode the four criteria guaranteeing that E is a $t-$spanning set of L:

Constraint (5.3d): Let p be a motion in $\mathcal{B}$ from $o \in \mathcal{O}$ to some vertex $j' \in L$. If $p \notin E_o$, then $y_p^o = 0$ by definition. Therefore (5.3d) requires that $x_{ij}^{o'} = 0$ for all $(i,j) \in S_p$ and $o' \in \mathcal{O}$ implying that for all pairs (i,j) such that p is the motion from i to j, the copy $(i,j)^{o'}$ cannot appear in the tree $T^{o'}$. Alternatively, if $y_p^o = 1$, then $x_{ij}^{o'}$ is free to take values 1 or 0 for any $(i,j) \in S_p$ and $o' \in \mathcal{O}$. Thus constraint (5.3d) encodes the Usable Edge Criteria.

Constraint (5.3e): Constraint (5.3e) takes a similar form to [23, Equation (3.7a)]. Note that $M_{ij}^o \geq 0$ for all $(i,j) \in L^2, o \in \mathcal{O}$. Indeed, $\forall t \geq 1$, $M_{ij}^o \geq c_{oi} + c_{ij} - c_{oj}$, and $c_{oi} + c_{ij} \geq c_{oj}$ by the triangle inequality. Replacing M_{ij}^o in (5.3e) yields

$$z_i^o + c_{ij} - z_j^o \leq (t c_{oi} + c_{ij} - c_{oj})(1 - x_{ij}^o). \tag{5.4}$$

If $x_{ij}^o = 1$, then $(i,j)^o$ is on the path in T^o to vertex j and (5.4) reduces to $z_j^o \geq z_i^o + c_{ij}$ which encodes the Cost Continuity Criteria. If, however, $x_{ij}^o = 0$, then (5.4) reduces to $z_i^o - z_j^o \leq t c_{oi} - c_{oj}$ which holds trivially by constraint (5.3f) and by noting that $z_j^o \geq c_{oj}, \forall j \in L$ by the triangle inequality.

Constraint (5.3g): Constraint (5.3g) together with constraint (5.3e) yield the Arborescence Criteria. Indeed for all $o \in \mathcal{O}$, by Theorem 2.5 of [62], T^o is an arborescence rooted at o if every vertex in T^o other than o has exactly one incoming edge, and T^o contains no cycles. The constraint (5.3g) ensures that every vertex in L', which is the set of all vertices in T^o other than those in $\mathcal{O}$, have exactly one incoming edge, while constraint (5.3e) ensures that T^o has no cycles. Indeed, suppose that a cycle existed in T^o, and that this cycle contained vertex $i \in L'$. Suppose that this cycle is represented as

$$i \to j \to \cdots \to k \to i.$$

Recall that it is assumed that the cost of any motion between two different configurations in $\mathcal{X}$ is strictly positive. Therefore, (5.3e) implies that $z_i^o < z_j^o$ for any $(i, j) \in L^2$. Therefore,

$$z_i < z_j < \cdots < z_k < z_i,$$

which is a contradiction.

5.2.2 Motion Planning With a MTSCS

The previous section illustrates how to compute a control set $E = \bigcup_{o \in \mathcal{O}} E_o$ given the tuple $(\mathcal{X}, L, \mathcal{O}, c)$. We have also presented – see Section 2.3.2 – a description of how such a control set may be used during an online search of the weighted directed graph $G_{\text{CF}} = (L, \bar{E}_{\text{CF}}, c)$ from start vertex $p_s \in L$ to goal vertex $p_g \in L$. While any graph-search algorithm could be used to compute minimal cost paths in the graph G_{CF}, in this section we propose one alternative: an A* variant, *Primitive A* Connect* (PrAC) for path computation in G_{CF}. The algorithm follows the standard A* Algorithm 1 closely but with some variations.

Weighted fScore: In the standard A* algorithm given in Algorithm 1, two costs are maintained for each vertex i: the *gScore*, $g(i)$ – called the *cost to get* – representing the current minimal cost to reach vertex i from the starting vertex p_s and the *fScore*, $f(i) = g(i) + h(i)$ – called the *cost to go* – which is the sum of the gScore and an estimated cost to reach the goal vertex p_g by passing through i given heuristic h. At each iteration, the current expanded vertex is the one that minimizes f over all vertices in an open set. Thus equal weight is placed on the *cost to get* to each vertex, $g(i)$, and the *cost to go* from this vertex to p_g, $h(i)$. We implement the tunable fScore from [91]: Given a value $\lambda \in [0, 1]$, we let $a = 0.5\lambda, b = 1 - 0.5\lambda$, and we define a new cost function $f' = ag(i) + bh(i)$. If $\lambda = 1$, then both cost to get and cost to go are weighted equally, resulting in the standard A* algorithm. However, as λ approaches 0, more weight is placed on the cost to go resulting in expanding vertices with smaller heuristic values. This promotes *exploration*

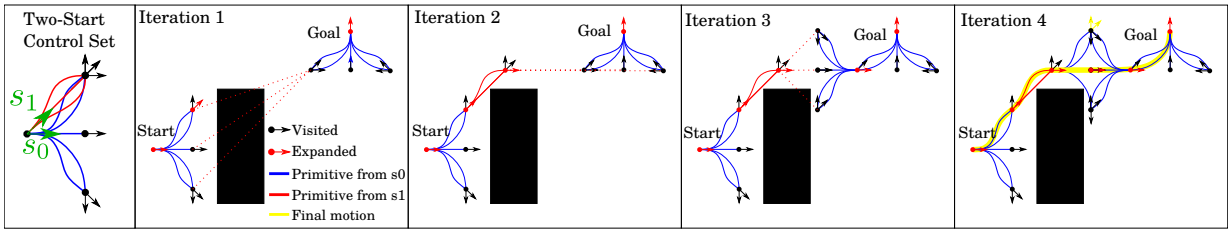


Figure 5.2: Example motion planning using PrAC for a 2-start lattice.

over *optimization*. While using a value $\lambda < 1$ eliminates optimality guarantees, it also empirically improves run-time performance.

Optimality may be guaranteed via the Anytime Repairing A* algorithm presented in [69] which begins with a small value of λ to quickly produce an initial solution and then increases λ towards 1 which improves upon their initial guess. While PrAC could be easily modified to adopt this framework, it has been found empirically that optimality is often reclaimed once the path is smoothed via the smoothing algorithm presented in the next section. In practice, using $\lambda = 1$ works well for maneuvers where p_s and p_g are close together, like parallel parking, while small values of λ work well for longer maneuvers like traversing a parking lot.

Observe that setting $\lambda = 0$ results in an algorithm which expands those vertices which maximally take the current vertex to the goal. This is a technique used often in motion planning on highways [119].

Expanding Start and Goal Vertices: We employ the bi-directional algorithm from [14, 114]. In detail, we expand vertices neighboring both start and goal vertices and attempt to connect these vertices on each iteration. In essence, we double the expansion routine at each iteration of the while loop in Line 7 of Algorithm 1: once from p_s to p_g and once from p_g to p_s with reverse orientation. We maintain two trees, one rooted at the start vertex p_s, denoted T_s, the other at the goal, T_g whose leaves represent open sets O_s, O_g, respectively. We also maintain two sets containing the current best costs $g(p_s, i)$ to get from p_s to each vertex $i \in T_s$, and from p_g to each vertex $j \in T_g$ (traversed in reverse), $g(p_g, j)$, respectively. Given an admissible heuristic h', we define a new heuristic h as

$$\begin{aligned} h(i) &= \min_{j \in O_g} h'(i,j) + g(p_g, j), \ \forall i \in O_s \\ h(j) &= \min_{i \in O_s} h'(j,i) + g(p_s, i), \ \forall j \in O_g \end{aligned} \tag{5.5}$$

On each iteration of the A* while loop in Line 7 of Algorithm 1, let $i \in O_s$ be the vertex that is to be expanded, let $j \in O_g$ be the vertex that solves (5.5) for i, and let p denote

the pre-computed motion from $i \in L$ to $j \in L$. We expand vertex i by applying available motions in $E_{R(i)} \cup \{p\}$ where $E_{R(i)}$ is the set of available motions at the relative start of i. The addition of p to the available motions improves the performance of the algorithm by allowing quick connections between T_s, T_g where possible. In the same iteration, we then swap T_s and T_g and perform the same steps but with all available motions in reverse. This is illustrated in Figure 5.2. Expanding start and goal vertices has proven especially useful during complex maneuvers like backing into a parking space.

Off-lattice Start and Goal Vertices: For a configuration $v \in \mathcal{X}$ and a lattice L given by (2.4), let Round(v) denote a function that returns the element of L computed by rounding each state of v to the closest discretized value of that state in L. For lattice L with start set S given by (2.5) and control set E obtained by solving the MILP in (5.3), it is likely that the start and goal configurations $p_s, p_g \in \mathcal{X}$ do not lie in L. This is particularly true in problems that necessitate frequent re-planning. While increasing the fidelity of the lattice or computing motions online between lattice vertices and p_s, p_g (e.g., [76]) can address this issue, both these methods adversely affect the performance of the planner. Instead we propose a method using lattices with graduated fidelity, a concept introduced in [90]. We compute a control set E_{off} of primitives from off-lattice configurations lattice vertices. These can be traversed in reverse to bring lattice vertices to off-lattice configurations. The technique is summarized in Algorithm 5 which is executed offline. The algorithm takes as input L, E, and a vector of tolerances for each state $Tol = (x_{\text{tol}}, y_{\text{tol}}, \theta_{\text{tol}}, u_{0,\text{tol}}, \ldots, u_{N,\text{tol}})$. It first computes a set $Q \subset \mathcal{X}$ such that for every configuration in $v \in \mathcal{X}$ there exists a configuration $q \in Q$ that can be translated to $q' \in \mathcal{X}$ where each state of q' is within the accepted tolerance of v (Line 2). For each element of Q, we determine the lattice vertex $q' = \text{Round}(q)$, and the set of available actions at the relative start of q', $E_{R(q')}$. For each primitive in $E_{R(q')}$, we compute a motion from q to a lattice vertex close to the endpoint of p and store it in a set E_q (Lines 7-13). In Lines 9, 10 the vertex p is modified to p' that is no closer to q than p. This is to ensure that the motion added to E_q in Line 13 does not posses large loops not present the primitive p. These loops can arise if the start and end configurations of a motion are too close together. An illustrative example of this principle is given in Figure 5.3.

Algorithm 5 Generate Off-Lattice Control Set

1: **procedure** OFFLATTICE(L, E, Tol)
2: $E_{\text{off}} \leftarrow \emptyset$

$$\begin{aligned} Q \leftarrow & \{2ix_{\text{tol}}, i = 0, \ldots, {}^{\alpha}/_{2x_{\text{tol}}}\} \\ & \times \{2iy_{\text{tol}}, i = 0, \ldots, {}^{\beta}/_{2y_{\text{tol}}}\} \\ & \times \{2i\theta_{\text{tol}}, i = 0, \ldots, {}^{(\pi-\theta_{\text{tol}})}/_{\theta_{\text{tol}}}\} \\ & \times \prod_{i=0}^{N} \{U_i^0 + 2ju_{i,\text{tol}}, j = 0, \ldots {}^{(U_i^1 - U_i^0)}/_{2u_{i\text{tol}}}\}. \end{aligned}$$

3: **for** $q \in Q$ **do**
4: $(x_q, y_q, \theta_q, u_{0,q}, \ldots, u_{N,q}) \leftarrow q$
5: $E_q \leftarrow \emptyset$
6: $q' \leftarrow \text{Round}(q)$
7: **for** $p \in E_{R(q')}$ **do**
8: $(x_p, y_p, \theta_p, u_{0,p}, \ldots, u_{N,p}) \leftarrow p$
9: **for** $x \in \{x_p, x_p + \text{sign}(x_p - x_q)\alpha\}$ **do**
10: **for** $y \in \{y_p, y_p + \text{sign}(y_p - y_q)\beta\}$ **do**
11: $p' \leftarrow (x, y, \theta_p, u_{0,p}, \ldots, u_{N,p})$
12: Compute motion from q to q'
13: Add motion of lowest cost over all p' to E_q
14: Add E_q to E_{off}
15: **return** E_{off}

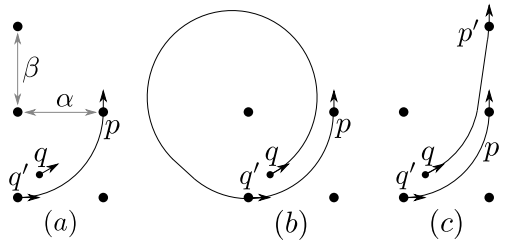


Figure 5.3: Algorithm 5 Lines 9-13 Example. (a) Configuration $q \in \mathcal{X} - L$, lattice vertex $q' = \text{Round(q)}$, and primitive $p \in E_{R(q')}$. (b) Motion from q to end of p results in a loop as q, p are too close together. (c) Vertex p replaced with neighboring vertex p' and a motion from q to p' is computed.

Theorem 5.2.3 (Completeness)**.** *If $\lambda = 1$, then PrAC returns the cost-minimizing path in $G_{CF} = (L, \bar{E}_{CF}, c)$.*

Proof. Given the assumptions of the Theorem, we may view PrAC as standard A* where the expansion of T_g serves only to update the heuristic. That is, PrAC reduces to standard A* with a heuristic given by (5.5) for all $i \in O_s$. At each iteration of A*, h is improved by expanding T_g. By the completeness of A* for admissible heuristics and finite branching factor (which is the case here for finite control sets), it suffices to show that h is indeed admissible. At each iteration of A*, let i be the vertex in O_s that is to be expanded. Let π be the optimal path from i to p_g in G_{CF}. Then π must pass through a vertex r in O_g. This result holds by the completeness of A* with initial configuration p_g with paths traversed in reverse. Then by the triangle inequality, $c(\pi) \geq c(\pi^{\bar{E}_{\text{CF}}}(i, r)) + g(p_g, r)$. Finally, because h' is an admissible heuristic, $c(\pi^{\bar{E}_{\text{CF}}}(i, r)) + g(p_g, r) \geq h'(i, r) + g(p_g, r) \geq \min_{j \in O_g} h'(i, j) + g(p_g, j) = h(i)$. Therefore, $h(i) \leq c(\pi)$ which concludes the proof. □

5.2.3 Motion Smoothing

Given the tuple $(\mathcal{X}, L, \mathcal{O}, c, E)$, of configuration space, lattice, start set, cost function, and MTSCS, respectively, let $G_{\text{CF}} = (L, \bar{E}_{\text{CF}}, c)$ be the weighted directed graph with edge set $\bar{E}_{\text{CF}}$ is the collision-free subset of $\bar{E}$ given in (2.7). We now present a smoothing algorithm based on the *shortcut* approach that takes as input a path in G_{CF}, here $\pi^E(p_s, p_g)$, between start and goal configurations p_s, p_g. This path is expressed as a sequence of edges in $\bar{E}_{\text{CF}}$. Thus, $\pi^E(p_s, p_g) = \{(i_r, i_{r+1}), r = 1, \ldots, m-1\}$ for some $m \in \mathbb{N}_{\geq 2}$ where $(i_r, i_{r+1}) \in \bar{E}_{\text{CF}}$ for all $r = 1, \ldots, m-1$ and $i_1 = p_s, i_m = p_g$. Algorithm 6 summarizes the proposed approach.

Algorithm 6 takes as input a collision-free path $\pi^E(p_s, p_g)$ between start and goal configurations in the graph G_{CF} – such as that returned by PrAC – as well as a set of obstacles $\mathcal{X}_{\text{obs}}$, and cost function of motions c. It also takes a function $\Psi : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ which is used to penalize reverse motion. That is, given two configurations $i, j \in \mathcal{X}$ with a motion p from i to j, we say that the cost of the motion p is $c(p)$, while the cost of the reverse motion p' from j to i that is identical to p but traversed backwards is $c(p') = \Psi(c(p))$.

The set V in Line 2 represents the set of all configurations in L that are endpoints of the edges $(i_r, i_{r+1}) \in \pi^E$. Critically, configurations in V are in the order in which they appear along π^E. In Lines 3-6, the distance to each vertex in V is set infinity except for $i_1 = p_2$ and the predecessor of i_1 is set to None. Next, the algorithm process each vertex $i_u, u = 1, \ldots, m-1$ along the path $\pi^E(p_s, p_g)$ and each vertex $i_v, v = u+1, \ldots m$ lying

Algorithm 6 Smoothing Lattice Motion

procedure DAGSMOOTH($\pi^E(p_s, p_g), \mathcal{X}_{\text{obs}}, c, \Psi$)
  $V = \{i_r\}_{r=1}^m$
  for $i \in V$ **do**
    $\text{dist}(i) = \infty$
  $\text{dist}(i_1) = 0$
  $\text{Pred}(i_1)$ =None
  for u from 1 to $m-1$ **do**
    for v from $u+1$ to m **do**
      $p_1 =$ motion from i_u to i_v
      $p_2 =$ motion from i_v to i_u
      $\underline{c} = \min(c(p_1), \Psi(c(p_2)))$, $\underline{p} = \arg\min(c(p_1), \Psi(c(p_2)))$
      $\overline{c} = \max(c(p_1), \Psi(c(p_2)))$, $\overline{p} = \arg\max(c(p_1), \Psi(c(p_2)))$
      if $\text{dist}(i_u) + \underline{c} \leq \text{dist}(i_v)$ **then**
        if CollisionFree($\underline{p}, \mathcal{X}_{\text{obs}}$) **then**
          $\text{dist}(i_v) = \text{dist}(i_u) + \underline{c}$
          $\text{Pred}(i_v) = i_u$
        else
          if $\text{dist}(i_u) + \overline{c} \leq \text{dist}(i_v)$ **then**
            if CollisionFree($\overline{p}, \mathcal{X}_{\text{obs}}$) **then**
              $\text{dist}(i_v) = \text{dist}(i_u) + \overline{c}$
              $\text{Pred}(i_v) = i_u$
  return Backwards chain of predecessors from i_m

farther along the path than i_u (Lines 7-8). For each such pair i_u, i_v, the algorithm attempts to connect i_u to i_v via either a forward motion p_1 from i_u to i_v with cost $c(p_1)$ or via a motion p_2 from i_v to i_u traversed backwards with cost $\Psi(c(p_2))$ (Lines 9-21). In particular, the algorithm first attempts to connect i_u to i_v using the motion in $\{p_1, p_2\}$ with least cost $\underline{c}$ (Lines 13-16). If this cannot be done without colliding with an obstacle (verified by the `CollisionFree` function), then the algorithm attempts to connect i_u to i_v using the other motion.

Were we to form a graph with vertices V and with edges $(i_u, i_v) \in V^2$ where $v > u$ and where the motion from i_u to i_v is does not collide with an obstacle, then this graph would be a directed, acyclic graph (DAG). Indeed, were this graph not acyclic, then the path $\pi^E(p_s, p_g)$ would contain a cycle implying that a configuration would appear at least twice in $\pi^E(p_s, p_g)$. This is impossible by construction of the PrAC algorithm which maintains two *trees* rooted at p_s, p_g, respectively (no cycles are present). This motivates the following observation and Theorem:

Observation 5.2.4. *The nested for loop in lines 7-8 of Algorithm 6 is constructing a directed, weighted, acyclic graph and solving the minimum path problem from p_s to p_g on that graph.*

Theorem 5.2.5. *Let π_1^E be the input path to Algorithm 6 between configurations p_s, p_g with cost $c(\pi_1^E)$. Let π_2^E be the path returned by Algorithm 6 with cost $c(\pi_2^E)$. Then $c(\pi_2^E) \leq c(\pi_1^E)$. Moreover, assuming that the function `CollisionFree` runs in constant time, Algorithm 6 runs in time quadratic in the number of vertices along π_1^E.*

Proof. By Observation 5.2.4, Algorithm 6 constructs a DAG containing configurations p_s, p_g as vertices. It also solves the minimum cost path problem on this DAG. Indeed, note that Lines 13-21 are identical to the shortest path in a DAG Algorithm 2. By construction of the DAG vertex set in Line 2, all configurations along the original path π_1^E are in V. Further, for all $r = 1, \ldots, m-1$, either the motion from i_r to i_{r+1} is collision free, or the motion from i_{r+1} to i_r traversed backwards is collision free since $\pi^E(p_s, p_g)$ is a collision-free path. Thus π_1^E is an available solution to the minimum path problem on the DAG. This proves that the minimum cost path can do no worse than $c(\pi_1^E)$. Finally, then V is the set of endpoints of edges in π_1^E. Therefore, $V \subseteq L$. Because all motions between lattice vertices have been pre-computed in $\mathcal{B}$, Lines 9, 10 can be executed in constant time, and the nested for loop in lines 7-8 will run in time $O(m)^2$ where m is the number of vertices on the path π_1^E. □

Algorithm 6 is similar to Algorithm 1 in [76] with one critical difference. The latter algorithm adopts a greedy approach, connecting the start vertex i_1 in the path π^E to

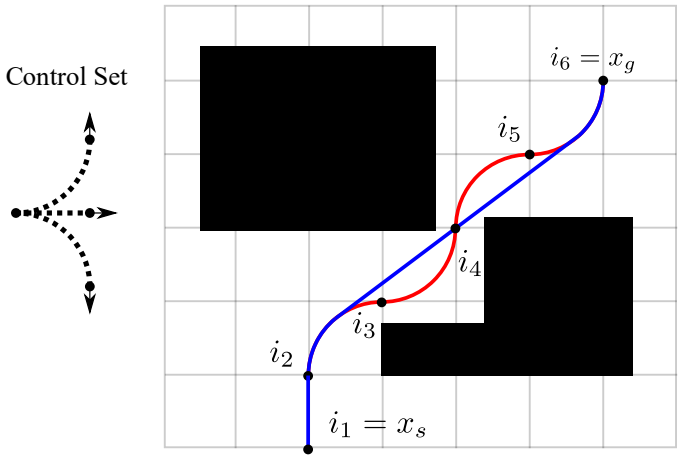


Figure 5.4: Comparison of smoothing algorithms for same input path (red). Blue: path smoothed using Algorithm 6. Red: path smoothed using Algorithm 1 from [76] (no change from input path).

the farthest vertex $i_k \in \pi^E$ that can be reached without collisions. This process is then repeated at vertex i_k until the goal vertex is reached. While this greedy approach also runs in time quadratic in the number of vertices along the input path, the cost of paths returned will be no lower than the cost of paths returned by Algorithm 6 proposed herein. A simple example is illustrated in Figure 5.4. Using the control set in Figure 5.4, an initial path is computed from $i_1 = p_s$ to $i_6 = p_g$ (red). This path remains unchanged when smoothed using the proposed Algorithm in [76]. However, Algorithm 6 will return the less costly blue path.

5.3 Evaluation

We verify our proposed technique against two state of the art techniques in two common navigational settings. All algorithms were encoded in Python 3.7 (Spyder). Results were obtained using a desktop equipped with an AMD Ryzen 3 2200G processor and 8GB of RAM running Windows 10 OS. Start and goal configurations were not constrained to be lattice vertices. We assume that all obstacles are known to the planner ahead of time and are stationary, and that the environment is noiseless. As such, the results that follow can be thought of as a single iteration of a full re-planning process. The workspace used in this section is $\mathcal{W} = \mathbb{R}^2$.

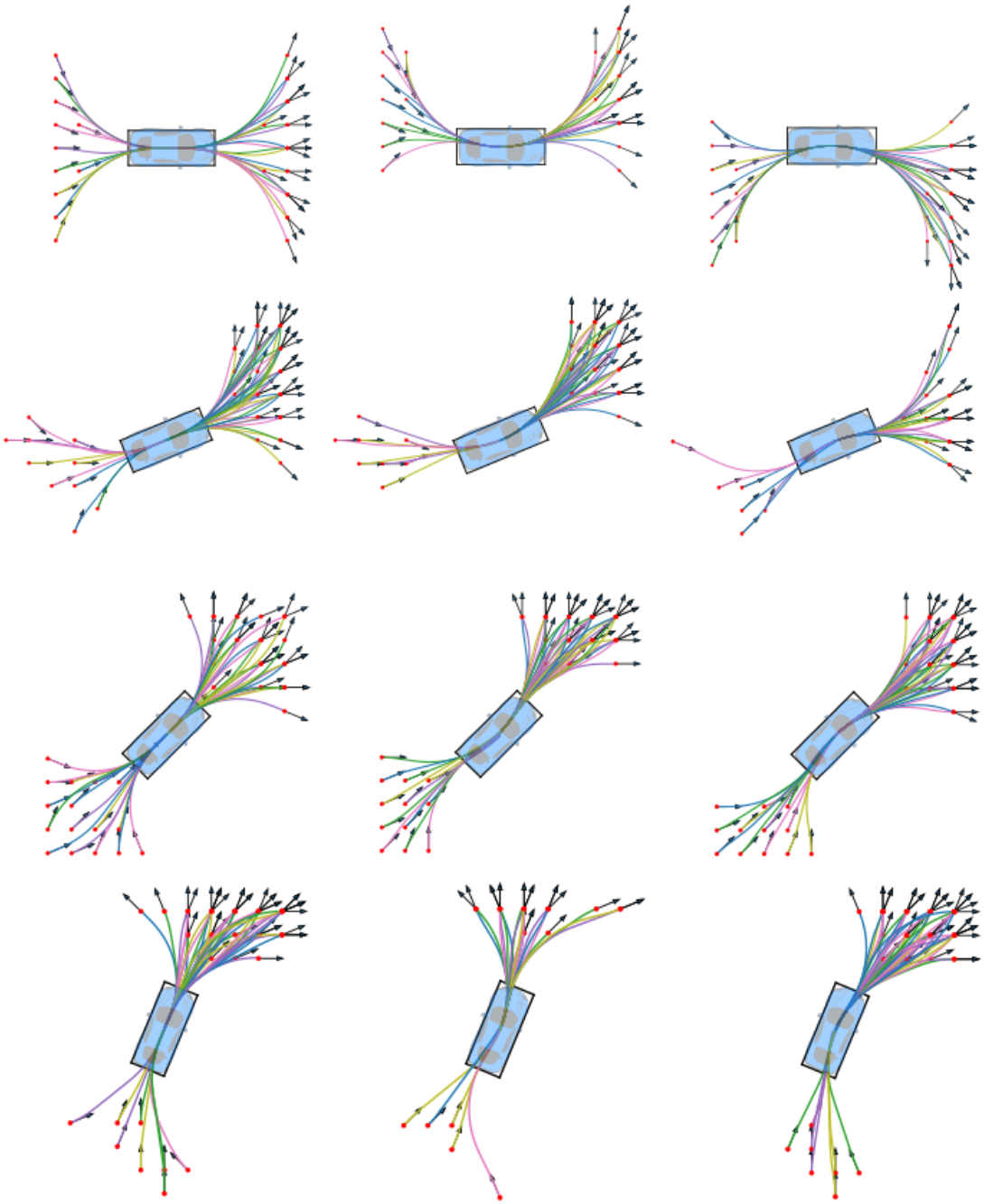


Figure 5.5: Motion primitives for each starting vertex.

5.3.1 Memory

The control set E we used was computed by solving the MILP (5.3) for a lattices described in the following sections. In each case, motions between lattice vertices were computed using the techniques outlined in Chapter 4. In section 5.3.2, we compute paths between start-goal configurations in parking lot scenarios. As such, we do not plan velocity profiles. As discussed in Chapter 4, in order to store a motion without a velocity profile, only three values must be saved to memory as the motion can be easily re-generated from these values. In Section 5.3.3, we compute full trajectories: paths and velocity profiles. In this case, trajectories can be easily generated using only 19 saved values (see Chapter 4).

5.3.2 Parking Lot Navigation

We begin by validating the proposed method against a common technique: Hybrid A* [25] in a parking lot scenario. Though Hybrid A* is not a new algorithm, more recent state of the art approaches use Hybrid A* to plan an initial motion which is then refined (e.g. [118]). We are therefore motivated to compare the run-time and path quality of the approach proposed in this chapter to Hybrid A* whose run-time is a lower bound of all state of the art algorithms using it as a sub-routine.

Lattice Setup & Pruning

The configuration space used here is $\mathcal{X} = \mathbb{R}^2 \times [0, 2\pi) \times \mathbb{R}^3$ with configurations $(x, y, \theta, \kappa, \sigma, \rho)$ representing position (x, y), heading θ, curvature κ, curvature rate σ, and the rate of change of curvature rate with respect to arc-length, ρ. Motion primitives were generated using the MILP in (5.3) for a 15×20 square lattice with 16 headings and 3 curvatures, and a value of $t = 1.1$ (10% error from optimal). Trajectories were generated using the methods outlined in Chapter 4 with weights $w_t = w_y = w_a = w_{\mathcal{J}} = 0.25$ representing an intermediate user. To account for the off-lattice start-goal pairs, we used a higher-fidelity lattice with 64 headings and 30 curvatures. Lattice vertex values of σ, ρ were set to 0. This results in a start set $\mathcal{O}$ with 12 starts given by (2.5). The cost c of lattice motions is given by (4.8) with unit speed. Bounds on κ, σ, ρ were set as:

$$\kappa_{\max} = 0.1982 m^{-1}, \ \sigma_{\max} = 0.1868 m^{-2} \ \rho_{\max} = 0.3905 m^{-3}. \tag{5.6}$$

which are considered comfortable for a user [6], particularly at low speeds typical of parking lots. The spacing of the lattice x, y-values was chosen to be $r_{\min}/4$ for a minimum turning

radius $r_{\min} = 1/\kappa_{\max}$. Finally, if the arc-length of the motion from $o \in \mathcal{O}$ to a vertex j was larger than 1.2 times the Euclidean distance from o to j for all $o \in \mathcal{O}$, then j was removed from the lattice. This technique which we dub *lattice pruning* is to keep the lattice relatively small, and to remove vertices for which the optimal motion requires a large loop. The value of 1.2 comes from the observation that the optimal motion from the start vertex $o = (0, 0, 0, \kappa_{\max})$ to $j = (r_{\min}, r_{\min}, \pi/2, \kappa_{\max})$ is a quarter circular arc of radius $r_{\min}$. The ratio of the arc-length of this maneuver to the Euclidean distance from o to j is $\pi/(2\sqrt{2}) \approx 1.11$. Thus using a cutoff value of 1.2 admits a sharp left and right quarter turn but is still relatively small.

Adding Reverse Motion

The motion primitives returned by the MILP in (5.3) are motions between a starting vertex $o \in \mathcal{O}$, and a lattice vertex $j \in L - \mathcal{O}$. As such, they are for forward motion only. To add reverse motion primitives to the control set E_o for each $o \in \mathcal{O}$, we reflect the states x, y, θ of each motion p starting at o about the line perpendicular to the heading of o while maintain the states κ, σ, ρ the result is a mirror image of the motion p that can be traverse backwards starting at o. We then rounded the final configurations of these primitives to the closest lattice vertex. For each $(x, y)-$value of the final configurations, we select a single configuration $(x, y, \theta, \kappa, \sigma, \rho)$ that minimizes arc-length. This is to keep the branching factor of an online search low. Finally, to each $o = (0, 0, \theta, \kappa, 0) \in \mathcal{O}$ we add three primitives to E_o: $(0, 0, \theta, \pm\kappa_{\max}, 0), (0, 0, \theta, 0, 0)$ with a reverse motion penalty. These primitives reflect the cars ability to stop and instantaneously change its curvature. The resulting primitives can be found in Figure 5.5

Scenario Results

We verified our results in four parking lot scenarios (a)-(e). The first four scenarios illustrate our technique in parking lots requiring forward and reverse parking. The results are illustrated in Figure 5.6. Here, we have compared out approach to Hybrid A* using an identical collision checking algorithm, and using the same heuristic (that proposed in [25]). Though the motions may appear similar, they are actually quite different. This difference is illustrated in Figure 5.7 which illustrates the heading θ of the vehicle along the motion proposed herein vs that of Hybrid A*. Heading profiles for the other scenarios are not included for brevity, though results are similar. To evaluate the quality of the motions predicted, we use three metrics: the integral of the square jerk (IS Jerk), final arc-length, and runtime. These three metrics are expressed as ratios of the value obtained using the

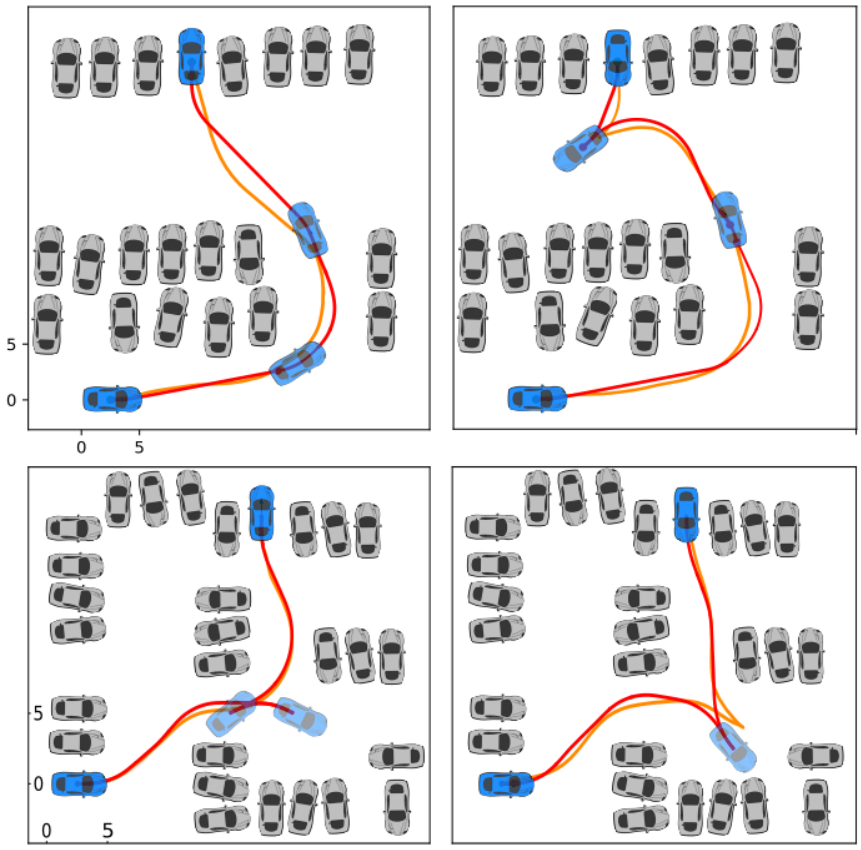


Figure 5.6: Scenarios (a) - (d). Red paths from proposed method, yellow from Hybrid A*.

methods of [25] to those of the proposed. The results are summarized in Table 5.1. The major difference between the two approaches can be seen in the IS Jerk Reduction. That is, the ratio of the IS Jerk using the methods of [25] to those of the proposed. This is due to the fact that the motion primitives we employ are each G^3 curves with curvature rates bounded by what is known to be comfortable. Observe that the value of IS Jerk obtained using our approach is up to 16 times less than that of [6]. In fact, using a Hybrid A* approach may result in motions with infeasibly large curvature rates resulting in larger tracking errors and increased danger to pedestrians.

Despite the bounds on curvature rate, the final arc-lengths of curves computed using our approach are comparable to those of Hybrid A*. Furhter, though Dubins' paths (which are employed by Hybrid A*) take on average two orders of magnitude less time to compute than G^3 curves, the runtime performance of our method often exceeds that of Hybrid A*. In

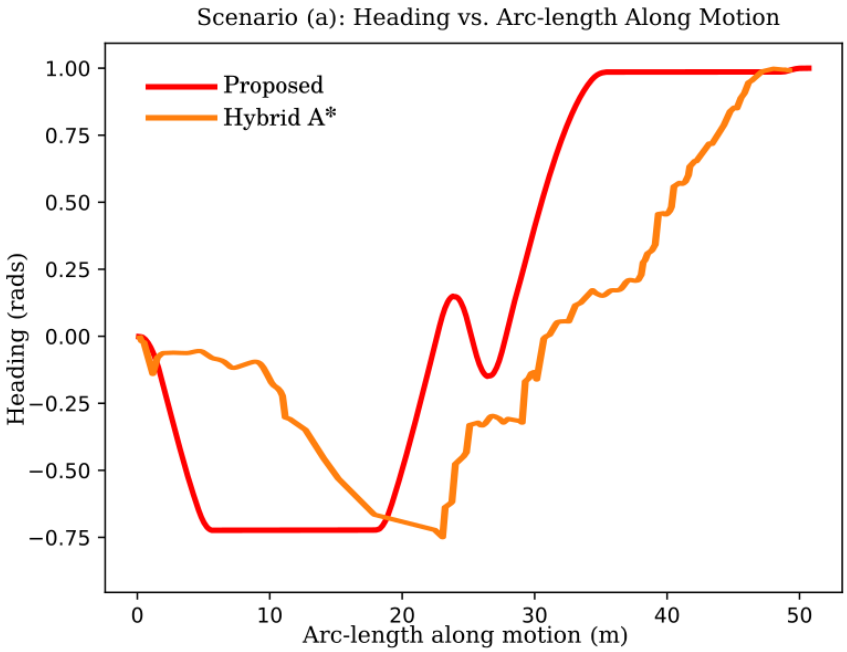


Figure 5.7: Heading along Motion for Scenario (a). Orange: Hybrid A* motion, Blue: proposed.

fact, our proposed method takes, on average 6.9 times less time to return a path, exceeding the average runtime speedup of the method proposed in [119]. Moreover, the methods in [119] do not account for reverse motion, and also assume that a set of way-points between start and goal configurations is known.

The only scenario in which Hybrid A* produces a motion in less time than the proposed method is Scenario (c) in which Hybrid A* produced a path with no reverse motion (which accounts for the speedup). However, in order to produce this motion, the curvature of the motion must change instantaneously multiple times resulting in an IS Jerk that is 16.3 times higher than the proposed method. The low run-time of the proposed method may due to the length of primitives we employ. It has been observed that Hybrid A* often takes several iterations to obtain a motion of comparable length to one of our primitives. This results in a much larger open set during each iteration of A*.

The final scenario we investigated is a parallel parking scenario (scenario (e)) which is illustrated in Figure 5.8. Though the motion computed with Hybrid A* may appear simpler, it requires a curvature rate that is 16.7 times larger than what is considered comfortable. It should also be noted that several other parallel parking scenarios in which

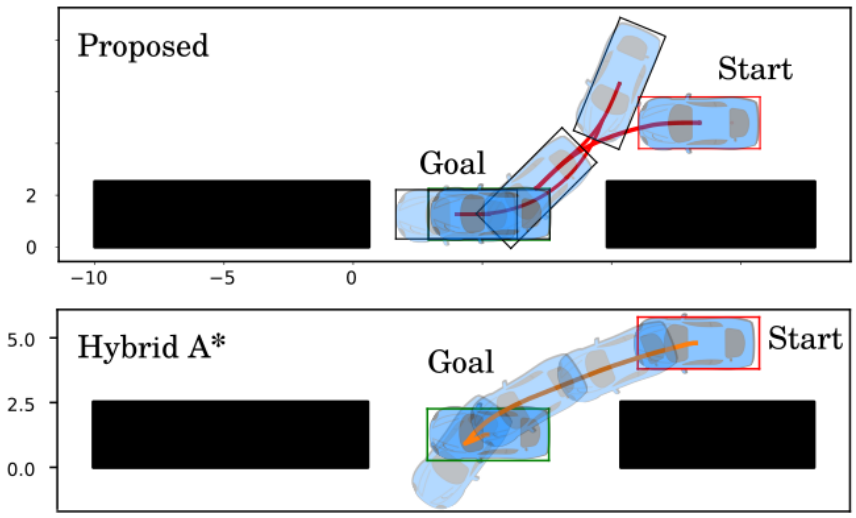


Figure 5.8: Scenario (e).

Scenario	IS Jerk Reduction	Length Reduction	Runtime Speedup
(a)	7.7	0.99	5.26
(b)	9.91	1.01	18.40
(c)	10.14	0.96	0.61
(d)	16.32	2.21	7.30
(e)	16.71	0.63	3.00

Table 5.1: Scenario Results

the clearance between obstacles was decreased. While the proposed method returned a path in each of these scenarios, Hybrid A* failed to produce a path in the allotted time.

A complex parking lot navigation scenario can be found in Figure 5.9. The solution uses the same motion primitives used in this section.

Figure 5.9: Motion planning using $t-$spanning G^3 motion primitives. Magenta: primitives used, Red: final motion, Cyan: car footprint.

5.3.3 Speed Lattice

In this experiment, we generate a full trajectory (including both path and speed profile) for use in highway driving using our approach. Here, we use only forward motion as reverse motion on a highway is unlikely.

In addition to developing G^3 paths, Chapter 4 also details a method with which a trajectory with configurations $(x, y, \theta, \kappa, \sigma, \rho, v, a, \beta)$ may be computed. Here, v, a, β represent velocity and longitudinal acceleration, and longitudinal jerk respectively. The approach is to compute profiles of ρ and β that result in a trajectory that minimizes a user-specified cost function. This cost function is a weighted sum of undesirable trajectory features including the integral of the square (IS) acceleration, IS jerk, IS curvature, and final arc-length. The key feature of this approach is that both path (tuned by ρ) and velocity profile (tuned by β) are optimized simultaneously, keeping path planning in-loop during the optimization. As in the previous set of examples, computing trajectories via the methods outlined in Chapter 4 require orders of magnitude more time than simple Dubins' paths.

However, pre-computing a set of motion primitives where each motion is itself computed using the methods of Chapter 4 ensures that every motion used in PrAC is optimal for the user. Moreover, because we include velocity in our configurations – and therefore in our primitives – we do not need to compute a velocity profile.

Lattice Setup & Pruning

Motion primitives were generated (5.3) for a 24×32 grid. Dynamic bounds for comfort were kept at (5.6). The x component of the lattice vertices were sampled every $r_{\min}/6$ meters while the y components were sampled every $r_{\min}/12$. Headings were sampled every $\pi/16$ radians (32 samples). We also assumed values of $\kappa = \sigma = \alpha = 0$ on lattice vertices. To account for off-lattice start-goal pairs, we use a higher-fidelity lattice with 128 headings, and 10 headings between $-\kappa_{\max,\kappa_{\max}}$. Finally, five evenly spaced velocities were sampled between 15 and 20 km/hr.

Scenario Results

The highway scenario was chosen to closely resemble the roadway driving scenario in [119]. The results of this sceario can be found in Figure 5.10, while performance analysis is summarized in Table 5.2. The metrics used to measure performance are the arc-length of the proposed motion, the smoothness cost of the motion, the maximum curvature obtained over the motion, and a runtime speedup normalized to Hybrid A* (HA*). The final column of the Table indicates weather a velocity profile was included during the motion computation. In Table 5.2, two values of smoothness cost are given in the form of a tuple $(\text{Smoothness}_1, \text{Smoothness}_2)$. The first value of the tuple refers to the definition of smoothness from [25]: sampling N configurations along a motion, we let $\mathbf{x}_i$ denote vector of x, y-components of the i^{th} configuration. Letting $\Delta\mathbf{x}_i = \mathbf{x}_i - \mathbf{x}_{i-1}$, the first metric of smoothness is given by:

$$\text{Smoothness}_1 = \sum_{i=1}^{N-1} |\Delta\mathbf{x}_{i+1} - \Delta\mathbf{x}_i|^2.$$

The second value of the tuple refers to the definition of smoothness used in [119]. Here, smoothness is given by

$$\text{Smoothness}_2 = \int_0^{s_f} \kappa(s)^2 ds,$$

the integral of the squared curvature along the motion to final arc-length s_f. The first two methods appearing in the Table are computed directly from the motions in Figure 5.10, while the second two come from [119] for an identical experiment. Here, CG refers to the method proposed in [119], while HA2* refers to the implementation of Hybrid A* as it appears in [119].

The authors of [119] report an average runtime speedup of 4.5 times as compared to Hybrid A* for the path planning phase (without speed profile). On average, PrAC

computed a full motion, including a speed profile 4.7 times faster than the time required for Hybrid A* to compute a path. Furthermore, the use of PrAC with G^3 motion primitives significantly reduced the smoothness cost in both of its possible definitions.

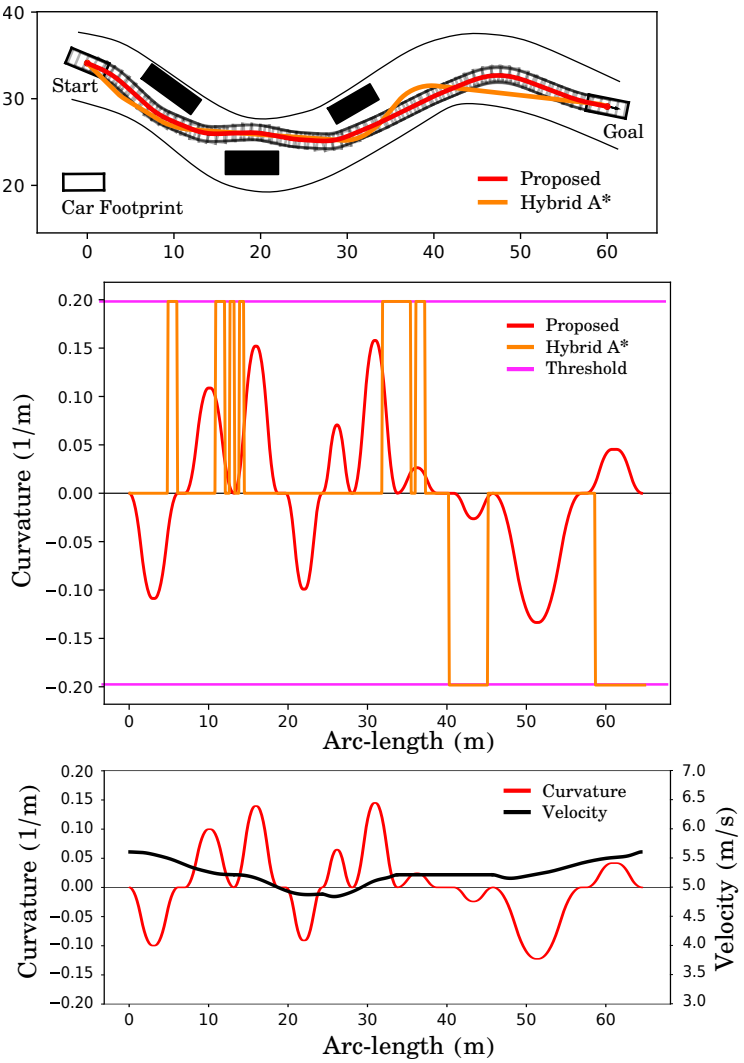


Figure 5.10: Result of highway maneuver with several obstacles.

Method	Length (m)	Smoothness Cost	Max Curvature (m^{-1})	Runtime Speedup	Velocity
HA*	65.0	(1.35, 0.77)	0.198	1	No
Proposed	64.6	(0.17, 0.28)	0.158	4.7	Yes
CG	65.8	(- , 0.44)	0.189	4.5	No
HA2*	65.6	(- , 0.88)	0.196	1	No

Table 5.2: Road navigation results: HA* and proposed shown above. CG and HA2* from [119] Table 3 for similar motion planning problem.

5.4 Discussion

The results of the previous section illustrate the effectiveness of the proposed technique. Indeed, feasible, smooth motions were computed between start and goal locations in both parking lot and highway settings. By adding reverse motion primitives, complex problems like navigating an obstacle-rich parking lot, or parallel parking were solved. Moreover, if the motion primitives already include velocity as a state, then a velocity profile may be easily computed. This work has not proposed a controller to track the reference paths we compute nor does it propose a framework for re-planning. These are left for future work. Further, lattices were constructed based on intuition and experimentation. In future, integrating formal lattice generation techniques may improve the quality of the resulting trajectories.

Chapter 6

Learning Control Sets For Lattice Planners From User Preferences

The research in this chapter was conducted in collaboration with Nils Wilde, another PhD. student at the University of Waterloo. This work was published in the proceedings of The 14th International Workshop on the Algorithmic Foundations of Robotics (WAFR) 2020 [13]. The authors A. Botros and N. Wilde contributed equally to the presented work.

6.1 Introduction

In Chapter 5, we illustrate how to compute a minimum $t-$spanning control set of a lattice given the tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W})$ of configuration space, lattice, start set, cost of vertex-to-vertex motions in L, and workspace, respectively. However, this approach assumes that the cost $c(p)$ for all motions between lattice vertices is known. In Chapter 4, we propose a technique to generate trajectories between vertices of a lattice that optimize a cost function that reflects the relative importance of travel time to comfort for a user. However, in Chapter 4, we assume that the weights representing the users preferences is known. Further, we assumed Assumption 2.2.2: that the cost function c for motions was additive.

In this chapter, we address the problem of *learning* motion primitives from user Demonstrations when the weights of the user are not known. This problem involves simultaneously learning the set $\bar{E}$ (given in (2.7)) of pairs of lattice vertices that should be connected by motion primitives, and the actual motions in E that connect lattice vertices in $\bar{E}$ in such

a way that is optimal for a user. An intuitive approach would be to apply a separation principle. In detail: for the tuple $(\mathcal{X}, L, \mathcal{O}, \mathcal{W})$,

1. Determine the cost function c of a single user from demonstrations.
2. Use the cost function c to compute the motions in $\mathcal{B}$, i.e., the motions from each start $o \in \mathcal{O}$ to each vertex $v \in L$ that solve the motion planning problem 2.2.1. This could be accomplished using, say, the techniques outlined in Chapter 4.
3. Select a control set $E \subseteq \mathcal{B}$ and its associated set of connecting vertices $\bar{E}$.

The main contribution of this Chapter is a proof that this intuitive approach is, in fact, an optimal solution to the problem of simultaneously learning connections $\bar{E}$ and their associated motions E given demonstrations from a single user. To address the third step of the afore mentioned process: the selection of the control set E, we propose to use a solution to the Minimum Spanning K-Control Set (MSKCS) problem 2.3.10. That is, we compute a control set $E = \bigcup_{o \in \mathcal{O}} E_o$ such that $\max_{o \in \mathcal{O}} |E_o|$ is bounded by an input value K. This in turn bounds the branching factor during an online search. The secondary contributions of this Chapter are a MILP reformulation of the MSKCS problem when the cost c is no longer assumed to be additive, and a proof of the NP-hardness of its decision version 2.3.11. This chapter relaxes the additive cost Assumption 2.2.2 slightly to include maximum values (this will be discussed in Assumption 6.2.1 which replaces Assumption 2.2.2 in this chapter).

To estimate solutions to the motion planning problem 2.2.1 for a user-specific cost, we consider that the quality of a motion is evaluated by a user who has preferences with respect to that motion. In autonomous driving, some users may prefer the vehicle to drive more aggressively, making sharp turns and maintaining high speeds, while others may prefer slower speeds and smoother turns [35, 37]. To find motions that reflect user preferences, we assume that users evaluate a vehicle's behaviour based on a weighted sum of features, similar to [35, 99].

The preferred behaviour of an autonomous vehicle often depends on situational context, like type of road or obstacles. Thus, a control set of motions that reflect the user preferences might only be applicable in certain scenarios. Complete planners that encompass many scenarios often use specialized lattices [1, 22, 35]. Our methodology is not limited to a specific scenario like driving on a highway or navigating in close proximity to an obstacle: for any of these scenarios we can learn a respective control set that captures the user preferences. Thus, we only consider global features such as travel time or maximum jerk, which are common in autonomous driving [35].

6.2 Problem Statement

In this chapter, we will be decoupling the problem of computing a control set E into a problem of computing a set of connections $\bar{E}$ from (2.7) and a motion realizing those connections. To this end, instead of using a given control set E to define the connections in $\bar{E}$ (as in (2.7)), we will be using a given $\bar{E}$ to define E. Since $\mathcal{O}$ is not assumed to have Property 2.3.1, care must be taken in defining E from $\bar{E}$. In this chapter, we assume that all $(i,j) \in \bar{E}$ are such that there exists $o \in \mathcal{O}$ and $j' \in L$ with $i \cdot p = j \implies o \cdot p = j'$. That is, no connections (i,j) are in $\bar{E}$ with motion p form i to j such that $i \cdot p$ is not valid.

Different users might have individual preferences for motions, which can be described by a respective cost function c. For each start $o \in \mathcal{O}$ and each configuration $j \in \mathcal{X}$, let p_{oj} be a motion from o to j. Suppose p_{oj} is the motion associated with the trajectory $\Pi_{ij}(t)$ that solves the motion planning problem 2.2.1 given a cost c_1 and $p_s = o, p_g = j$. Though p_{oj} may be the optimal motion from o to j for a user whose user-specific cost is given by c_1, it may be sub-optimal for a user with cost $c_2 \neq c_1$. We model users who evaluates a motion p_{oj} from $o \in \mathcal{O}$ to $j \in \mathcal{X}$ as a weighted sum of features $\boldsymbol{\phi}(p_{oj}) = \begin{bmatrix}\phi_1(p_{oj}) & \phi_2(p_{oj}) & \dots & \phi_n(p_{oj})\end{bmatrix}$:

$$c(p_{oj}, \boldsymbol{w}) = \boldsymbol{w} \cdot \boldsymbol{\phi}(p_{oj}), \tag{6.1}$$

where $\boldsymbol{w}$ is a vector of weights $\begin{bmatrix}w_1 & w_2 & \dots & w_n\end{bmatrix}$. Similar user cost functions have been used in [2, 74, 79, 99]. The features are assumed to take non-negative real values and can represent properties such as travel time, integral of the squared magnitude of jerk, maximum curvature, maximum acceleration, etc. Without loss of generality, we assume that $\boldsymbol{w} \in [0,1]^n$. Given a user weight $\boldsymbol{w}$, we denote the cost of the optimal motion from any $o \in \mathcal{O}$ to $j \in \mathcal{X}$ with respect to the weights $\boldsymbol{w}$ and the optimal motion as

$$c^*_{oj}(\boldsymbol{w}) = \min_{p_{oj}} \ \boldsymbol{w} \cdot \phi(p_{oj}), \ p^*_{oj}(\boldsymbol{w}) = \arg\min_{p_{oj}} \ \boldsymbol{w} \cdot \phi(p_{oj}), \tag{6.2}$$

respectively. Using these definitions, let $\mathcal{B}^{\boldsymbol{w}}$ be the set $\mathcal{B}$ given in (2.6) where motions p from $o \in \mathcal{O}$ to lattice vertices $j \in L - \mathcal{O}$ are given by $p^*_{oj}(\boldsymbol{w})$. That is, $\mathcal{B}^{\boldsymbol{w}}$ is the set of all optimal motions given $\boldsymbol{w}$ from all starts $o \in \mathcal{O}$ to all lattice vertices $j \in L - \mathcal{O}$. If $\mathcal{O}$ has Property 2.3.1, then for each pair of lattice vertices $i, j \in L$, $j \notin \mathcal{O}$ there exists a start $o \in \mathcal{O}$ such that the optimal motion from i to j given weights $\boldsymbol{w}$ is equal to the optimal motion $p^*_{oj'}$ from o to some vertex $j' \in L - \mathcal{O}$. That is, the optimal motion from i to j given weights $\boldsymbol{w}$ is an element of $\mathcal{B}^{\boldsymbol{w}}$ if $\mathcal{O}$ has Property 2.3.1. Otherwise, only those pairs (i,j) such that there exists $o \in \mathcal{O}, j' \in L$ with $i \cdot p = j \implies o \cdot p = j'$ will exhibit this behavior.

Next, let $\bar{E} \subseteq L \times (L - \mathcal{O})$ be a set of connections. Then, given weights $\boldsymbol{w} \in [0,1]^n$, we may define the associated control set given weights $\boldsymbol{w}$ as

$$E^{\boldsymbol{w}} = \{p^*_{oj}(\boldsymbol{w}) : (o,j) \in \bar{E}\}. \tag{6.3}$$

Observe that $E^{\boldsymbol{w}}$ is the set optimal motions given $\boldsymbol{w}$ between all lattice vertices i, j such that there exists $(o, j') \in \bar{E}$ with $i \cdot p^*_{oj'} = j$.

A set of connections $\bar{E}$ induces an unweighted directed graph $G^{\bar{E}} = (L, \bar{E})$. Let $P^{\bar{E}}_{oj}$ be *any* path in the graph $G^{\bar{E}}$ from a start $o \in \mathcal{O}$ to $j \in L$. Then for each edge e in the path $P^{\bar{E}}_{oj}$, e must lie in $\bar{E}$ implying that there is an associated motion in $E^{\boldsymbol{w}}$ connecting the endpoints of e. Therefore, the path $P^{\bar{E}}_{oj}$ defines a compound motion from o to j comprised of sub-motions in $E^{\boldsymbol{w}}$.

Let $(p_1, p_2, \dots)$ be the sequence of motions associated with the edges $(e_1, e_2, \dots)$ in a path $P^{\bar{E}}_{oj}$. We extend the cost function c for single motions to a cost function u of paths: For each feature ϕ_l there exists a feature function, let

$$f_l(P^{\bar{E}}_{oj}) = f_l(\{\phi_l(p_1), \phi_l(p_2), \dots\}). \tag{6.4}$$

Here f_l depends of the motions $(p_1, p_2, \dots)$ associated with the edges $(e_1, e_2, \dots) \in P^{\bar{E}}_{oj}$. We consider only a class of feature functions. Explicitly, we make the following assumption:

Assumption 6.2.1 (Additive or Maximizing Feature Functions). For a graph $G^{\bar{E}}$, let $P = \{e_1, e_2, \dots, e_r\}$ be a path with edges $e_i \in \bar{E}$ and $P_1 = \{e_1, \dots, e_i\}, P_2 = \{e_{i+1}, \dots, e_r\}$. We assume that for each feature ϕ_l, the feature function f_l is either

$$\begin{aligned} \textit{additive}\text{: } & f_l(\{\phi_l(P_1), \phi_l(P_2)\}) = \phi_l(P_1) + \phi_2(P_l), \text{ or} \\ \textit{maximizing}\text{: } & f_l(\{\phi_l(P_1), \phi_l(P_2)\}) = \max(\phi_l(P_1), \phi_2(P_l)). \end{aligned}$$

Additive or maximizing feature functions encompass a wide range including the integral of the squared magnitude of jerk, travel time, maximum acceleration, etc.

The user cost function of a path in the graph $G^{\bar{E}}$ is then given by the weighted sum of all features

$$u(P^{\bar{E}}_{oj}, \boldsymbol{w}) = w_1 f_1(P^{\bar{E}}_{oj}) + \cdots + w_n f_n(P^{\bar{E}}_{oj}). \tag{6.5}$$

Note that u is a generalization of the cost function c, i.e., c evaluates a single motion while u is a function of a set of motions whose associated edges form a path in $G^{\bar{E}}$. Note further that for any motion p – not just paths $P^{\bar{E}}_{oj}$ – we can evaluate a user-cost $u(p, \boldsymbol{w})$ using (6.5) by evaluating the features and feature functions for p. The user cost function u is

similar to a reward function in reinforcement learning. However, it is challenging for users – especially non-experts – to specify weights for such a reward function [99]. Summarizing all weights in a row vector $\boldsymbol{w}$ and all features in a column vector $\boldsymbol{f}(P^{\bar{E}}_{oj})$ allows us to write $u(P^{\bar{E}}_{oj}, \boldsymbol{w}) = \boldsymbol{w}\boldsymbol{f}(P^{\bar{E}}_{oj})$. Given weights $\boldsymbol{w}$, we define the optimal path in the graph $G^{\bar{E}}$ from a starting vertex $o \in \mathcal{O}$ to $j \in \mathcal{X}$ as

$$\pi^{\bar{E},\boldsymbol{w}}(o, j) = \arg\min_{P^{\bar{E}}_{oj}} u(P^{\bar{E}}_{oj}, \boldsymbol{w}). \tag{6.6}$$

Thus $\pi^{\bar{E},\boldsymbol{w}}(o, j)$ is an extension of the *path using* E from Definition 2.3.4 to include weights and non-additive cost.

Given a set of connections $\bar{E}$, and weights $\boldsymbol{w}$, we observe that the control set $E^{\boldsymbol{w}}$ may be defined using (6.3). Therefore, to construct a control set it suffices to compute the tuple $(\bar{E}, \boldsymbol{w})$. Let $\bar{\mathcal{B}}$ be the connection set associated with $\mathcal{B}^{\boldsymbol{w}}$. That is, $\bar{\mathcal{B}}$ is the set of all pairs $(i, j) \in L^2$ such that there exists a motion $p^*_{oj'}(\boldsymbol{w}) \in \mathcal{B}^{\boldsymbol{w}}$ with $i \cdot p^*_{oj'}(\boldsymbol{w}) = j$. Therefore, based on our cost function, the user-optimal behavior between all connections in $\bar{\mathcal{B}}$ can be described by a control set given by the tuple $(\bar{\mathcal{B}}, \boldsymbol{w}^*)$ where $\boldsymbol{w}^*$ denotes the *true user weights*. That is, user-optimal behavior is achieved between all connections $(i, j) \in \bar{\mathcal{B}}$ by selecting a control set given by the tuple of all connections $\bar{\mathcal{B}}$, and perfect knowledge of the true user weight $\boldsymbol{w}^*$. Users cannot provide $\boldsymbol{w}^*$ directly, and we learn about $\boldsymbol{w}^*$ from demonstrations. The true user weights may be dependent on the situation (e.g. highway vs city driving). This work focuses on a single situation but could be extended to account for multiple situations with the result being several control sets, one for each situation. For a given situation, we treat $\boldsymbol{w}^*$ as a hidden parameter which we have to estimate.

Consider a path $\pi^{\bar{E},\boldsymbol{w}_1}(o, j)$ from a start $o \in \mathcal{O}$ that is optimal for a control set $(\bar{E}, \boldsymbol{w}_1)$, and a second weight $\boldsymbol{w}_2$. The cost of path $\pi^{\bar{E},\boldsymbol{w}_1}$ can be *evaluated* by $\boldsymbol{w}_2$, which we write as:

$$u_{oj}(\bar{E}, \boldsymbol{w}_1 | \boldsymbol{w}_2) = \boldsymbol{w}_2 \cdot \boldsymbol{f}(\pi^{\bar{E},\boldsymbol{w}_1}(o, j)) = u(\pi^{\bar{E},\boldsymbol{w}_1}(o, j), \boldsymbol{w}_2). \tag{6.7}$$

We read this as *the cost from o to j using control set $(\bar{E}, \boldsymbol{w}_1)$, evaluated by weights $\boldsymbol{w}_2$, i.e., the cost of the path that is optimal given weights $\boldsymbol{w}_1$, for a user whose weights are $\boldsymbol{w}_2$.* This concept is similar to the regret of optimization problems [56]. Based on this notation, we can now pose the main problem statement:

Problem 6.2.2 (User control set)**.** Given the tuple $(\mathcal{X}, L, \mathcal{O}, \mathcal{W})$ – of configuration space, lattice, start set, and workspace, respectively – hidden user preferences $\boldsymbol{w}^*$, and a budget on the allowable branching factor, $K \in \mathbb{Z}_{>0}$, find a control set given by $(\bar{E}, \boldsymbol{w})$, with

$\bar{E} = \bigcup_{o\in\mathcal{O}} \bar{E}_o \subseteq \bar{\mathcal{B}}, \boldsymbol{w} \in [0,1]^n$ such that

$$(\bar{E}, \boldsymbol{w}) = \arg\min_{\bar{E}', \boldsymbol{w}'} \max_{\substack{j\in L-\mathcal{O} \\ o\in\mathcal{O}}} \frac{u_{oj}(\bar{E}', \boldsymbol{w}'|\boldsymbol{w}^*)}{c^*_{oj}(\boldsymbol{w}^*)} \quad s.t. \max_{o\in\mathcal{O}} |\bar{E}'_o| \leq K, \ \boldsymbol{w}' \in [0,1]^n. \tag{6.8}$$

The control set $(\bar{E}, \boldsymbol{w})$ minimizes the maximum ratio of path costs *evaluated* by $\boldsymbol{w}^*$ to the cost of the optimal direct trajectory from o to j, given a budget of K connections. In essence $(\bar{E}, \boldsymbol{w})$ is the control set with the minimal t-error and is therefore the most robust control set. This motivates this motivates the following observation:

Observation 6.2.3. *Given weights $\boldsymbol{w}$ and a connection set $\bar{E}$, let $E^{\boldsymbol{w}}$ be given in (6.3). Further, consider the tuple $(\mathcal{X}, L, \mathcal{O}, u(\cdot, \boldsymbol{w}), \mathcal{W})$ where the cost of motions p between lattice vertices is given by $u(p, \boldsymbol{w})$ from (6.5). Then the $t-$error given in (2.3.6) for the control set $E^{\boldsymbol{w}}$ is*

$$tEr(E^{\boldsymbol{w}}) = \max_{\substack{o\in\mathcal{O} \\ j\in L-\mathcal{O}}} \frac{u_{oj}(\bar{E}, \boldsymbol{w}|\boldsymbol{w})}{c^*_{oj}(\boldsymbol{w})}.$$

Indeed, from (6.7), (6.6), and Definition 2.3.5:

$$u_{oj}(\bar{E}, \boldsymbol{w}|\boldsymbol{w}) = u(\pi^{\bar{E},\boldsymbol{w}}(o,j), \boldsymbol{w}) = d^{E^{\boldsymbol{w}}}(o,j). \tag{6.9}$$

The last inequality holds because $\pi^{\bar{E},\boldsymbol{w}}(o,j)$ the minimum cost path from o to j in the graph whose edges are those tuples in $\bar{E}$ and whose cost is $u(\cdot, \boldsymbol{w})$. This is precisely the definition of the distance using $E^{\boldsymbol{w}}$ form o to j. Further, $c^*_{oj}(\boldsymbol{w})$ is defined in (6.2) as the cost of the optimal direct motion from o to j given w. Thus, $c^*_{oj}(\boldsymbol{w}) = u_{oj}(\bar{\mathcal{B}}, \boldsymbol{w}|\boldsymbol{w})$ where $\bar{\mathcal{B}}$ is the set of all connections $(i,j) \in L \times (L - \mathcal{O})$ – including the direct connection from o to j. Therefore, $c^*_{oj}(\boldsymbol{w}) = u_{oj}(\bar{\mathcal{B}}, \boldsymbol{w}|\boldsymbol{w}) = d^{\mathcal{B}^w}(o,j)$, and (6.9) reduces to the definition of the $t-$error in Definition 2.3.6 for a lattice with vertex-to-vertex costs $u(\cdot, \boldsymbol{w})$.

6.3 Approach

Problem 6.2.2 consists determining a control set $(\bar{E}, \boldsymbol{w})$ that minimizes the maximum ratio between path cost and optimal cost. We introduce a model for how users provide demonstrations and then analyse how the unknown parameter $\boldsymbol{w}^*$ in equation (6.8) can be substituted by an estimate of the user weights given data. We show that the optimal solution is a pair $(\bar{E}, \boldsymbol{w})$ where $\boldsymbol{w}$ is the best available estimate of $\boldsymbol{w}^*$ and $\bar{E}$ can be computed via a mixed-integer linear program. This allows for a simple, yet effective method for solving Problem 6.2.2.

6.3.1 User model

We consider a user with a preference $\boldsymbol{w}^*$. Let d be a demonstrated motion from a start $o \in \mathcal{O}$ to a goal configuration $j \in X$. We do not require j to be a lattice vertex as we can still evaluate features $\boldsymbol{f}(d)$ and thus assign a cost $u(d, \boldsymbol{w})$ as in (6.5). We denote $d^{\boldsymbol{w}^*}$ the minimal-cost demonstration from o to j given weights $\boldsymbol{w}^*$. Users cannot perfectly demonstrate $d^{\boldsymbol{w}^*}$. Thus we use the features of demonstrations to formulate a probabilistic user model:

Assumption 6.3.1 (User Model). A user with a preference $\boldsymbol{w}^*$ provides a demonstration d from o to j with features $\boldsymbol{f}(d)$ where the density $p(\boldsymbol{f}(d)|\boldsymbol{w}^*)$ is:

$$p(\boldsymbol{f}(d)|\boldsymbol{w}^*) \sim \mathcal{N}(\boldsymbol{f}(d^{\boldsymbol{w}^*}), \boldsymbol{\sigma}). \tag{6.10}$$

Similar user models have been used in [74, 83]. Thus, users provide demonstrations such that the features follow a normal distribution centred at the features of the optimal demonstration. That is the user demonstrations are unbiased and, given a large enough data set, the average features of demonstrations equal the optimal features. Following (6.5), two demonstrations with equal features have the same cost for any user and thus are indistinguishable with respect to the cost function. Hence, we consider only features of demonstrations. Let $\mathcal{D} = (d_1, d_2, \dots)$ be a sequence of demonstrations. We then find the conditional expectation of $\boldsymbol{w}$ given $\mathcal{D}$ by taking the Bayesian posterior:

$$\mathbb{E}_{\boldsymbol{w}}[\boldsymbol{w}|\mathcal{D}] = \frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}} \mathbb{E}[\boldsymbol{w}|\boldsymbol{f}(d)]p(\boldsymbol{f}(d)) = \frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}} \int_{\boldsymbol{w}\in[0,1]^n} \boldsymbol{w}\ p(\boldsymbol{f}(d)|\boldsymbol{w})\ p(\boldsymbol{w}) d\boldsymbol{w}. \tag{6.11}$$

Finally, we can approximate the integral by summing over a set of N samples:

$$\mathbb{E}_{\boldsymbol{w}}[\boldsymbol{w}|\mathcal{D}] \approx \frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}} \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{w}_i\ p(\boldsymbol{f}(d)|\boldsymbol{w}_i)\ p(\boldsymbol{w}_i). \tag{6.12}$$

6.3.2 Estimation of the loss function

We now use the user model to find a solution to Problem 6.2.2, given a set of user demonstrations $\mathcal{D}$. We consider two approaches. The first is taking the conditional expectation over equation (6.8), given $\mathcal{D}$:

$$\begin{aligned}(\bar{E}, \boldsymbol{w}) = \arg\min_{\bar{E}', \boldsymbol{w}'} \mathbb{E}_{\boldsymbol{w}} \left[\max_{\substack{j\in L-\mathcal{O} \\ o\in\mathcal{O}}} \frac{u_{oj}(\bar{E}', \boldsymbol{w}'|\boldsymbol{w}^*)}{c^*_{oj}(\boldsymbol{w}^*)} \Big| \mathcal{D} \right] \\ s.t.\ \max_{o\in\mathcal{O}} |\bar{E}'_o| \leq K,\ \boldsymbol{w}' \in [0,1]^n.\end{aligned} \tag{6.13}$$

The second approach is to use the expectation $\hat{\boldsymbol{w}} = \mathbb{E}[\boldsymbol{w}|\mathcal{D}]$ to compute $(\bar{\bar{E}}, \boldsymbol{w})$, also known as the plug-in estimator [111]:

$$\begin{aligned}(\bar{E}, \boldsymbol{w}) = \arg\min_{\bar{E}', \boldsymbol{w}'} \max_{\substack{j \in L - \mathcal{O} \\ o \in \mathcal{O}}} \frac{u_{oj}(\bar{E}', \boldsymbol{w}'|\hat{\boldsymbol{w}})}{c^*_{oj}(\hat{\boldsymbol{w}})} \\ s.t. \max_{o \in \mathcal{O}} |\bar{E}'_o| \leq K, \ \boldsymbol{w}' \in [0,1]^n.\end{aligned} \tag{6.14}$$

While it is an approximation, (6.14) approaches the desired Problem 6.2.2 as $|\mathcal{D}| \to \infty$. Thus, this work focuses on solving (6.14).

6.3.3 Main Results

In this section, we present the main theorem of this paper that proposes a solution to the minimization problem in (6.14). The high-level idea is: given demonstrations $\mathcal{D}$, the expected user weight $\hat{\boldsymbol{w}}$ is computed. This user weight is used to calculate motions that minimize the user cost for all connections in $\bar{\mathcal{B}}$. Finally, a set $\bar{E} = \bigcup_{o \in \mathcal{O}} \bar{E}_o \subseteq \bar{\mathcal{B}}$ such that $\max_{o \in \mathcal{O}} |\bar{E}_o| \leq K$ is found to produce a control set $(\bar{E}, \hat{w})$. We make an observation about the cost function u, that motivates our main results.

Observation 6.3.2 (Weight Choice)**.** *For any set of connections $\bar{E}$, any vertices $o \in \mathcal{O}$, $j \in L - \mathcal{O}$, and any pair of weights $\boldsymbol{w}, \boldsymbol{w}' \in [0,1]^n$, it must hold that*

$$u_{oj}(\bar{E}, \boldsymbol{w}|\boldsymbol{w}) \leq u_{oj}(\bar{E}, \boldsymbol{w}'|\boldsymbol{w}). \tag{6.15}$$

Indeed, from the definition of paths $\pi^{\bar{E}, \boldsymbol{w}}(o, j)$ in (6.6), and the definition of user costs evaluated by other weights in (6.7), we observe that

$$u_{oj}(\bar{E}, \boldsymbol{w}|\boldsymbol{w}) = u(\pi^{\bar{E}, \boldsymbol{w}}(o, j), \boldsymbol{w}) \leq u(\pi^{\bar{E}, \boldsymbol{w}_1}(o, j), \boldsymbol{w}) = u_{oj}(\bar{E}, \boldsymbol{w}'|\boldsymbol{w}).$$

Intuitively, this observation is saying that a path from o to j that is optimal for a user will be evaluated as more favorable for that user than a path from o to j that is optimal for a user with different weights.

Theorem 6.3.3 (Problem Solution)**.** *The tuple $(\bar{E}, \bar{\boldsymbol{w}})$ is a solution to minimization problem (6.14) if and only if $\bar{\boldsymbol{w}} = \hat{\boldsymbol{w}}$, and $\bar{E}$ is such that $E^{\hat{\boldsymbol{w}}}$ (given in (6.3)) is a solution to the MSKCS problem 2.3.10 with input $(\mathcal{X}, L, \mathcal{O}, u(\cdot, \hat{\boldsymbol{w}}))$ and K. Here, the cost of a motion p between lattice vertices is given by $u(p, \hat{\boldsymbol{w}})$ in (6.5).*

Proof. Observe that Observation 6.3.2 implies that the optimal value for $\bar{\boldsymbol{w}}$ in equation (6.14) is given by $\hat{\boldsymbol{w}}$ for any set of connections $\bar{E}'$. Indeed, for any weights $\boldsymbol{w}' \in [0,1]^n$, equation (6.15) implies that $u_{oj}(\bar{E}', \boldsymbol{w}'|\hat{\boldsymbol{w}}) \geq u_{oj}(\bar{E}', \hat{\boldsymbol{w}}|\hat{\boldsymbol{w}})$. Thus, for any $\bar{E}'$, the pair $(\bar{E}', \boldsymbol{w}')$ must result in paths whose cost is at least $(\bar{E}', \hat{\boldsymbol{w}})$. This allows us to simplify equation (6.14) to

$$\begin{aligned} \bar{E} = \arg\min_{\bar{E}'} \max_{\substack{j \in L - \mathcal{O} \\ o \in \mathcal{O}}} & \frac{u_{oj}(\bar{E}', \hat{\boldsymbol{w}}|\hat{\boldsymbol{w}})}{c^*_{oj}(\hat{\boldsymbol{w}})}. \\ s.t. \max_{o \in \mathcal{O}} & |\bar{E}'_o| \leq K. \end{aligned} \tag{6.16}$$

By Observation 6.2.3, this last reduces to

$$\begin{aligned} \bar{E} = \arg\min_{\bar{E}'} & t\mathrm{Er}(E'^{\hat{\boldsymbol{w}}}). \\ s.t. \max_{o \in \mathcal{O}} & |\bar{E}'_o| \leq K. \end{aligned}$$

where $E'^{\hat{\boldsymbol{w}}}$ is the control set associated with $\bar{E}'$ by (6.3) given weights $\hat{\boldsymbol{w}}$. This last is exactly the MSKCS problem 2.3.10. Therefore, $(\bar{E}, \bar{\boldsymbol{w}})$ if and only if $\bar{\boldsymbol{w}} = \hat{\boldsymbol{w}}$. Further, since $\bar{\boldsymbol{w}} = \hat{\boldsymbol{w}}$ the set of connections $\bar{E}$ solves the simplified problem (6.16) if and only if its associated control set $E^{\hat{\boldsymbol{w}}}$ is a MSKCS of the lattice whose vertex-to-vertex costs is given by $u(\cdot, \hat{\boldsymbol{w}})$. □

Corollary 6.3.4. *If the control set $(\bar{E}, \boldsymbol{w})$ is a solution to equation (6.8), then $\boldsymbol{w} = \boldsymbol{w}^*$, and $\bar{E}$ is a MSKCS on the lattice L where costs of vertex-to-vertex motions p is given by $u(p, \boldsymbol{w}^*)$.*

The proof of Corollary (6.3.4) follows closely the proof of Theorem 6.3.3, and is therefore omitted.

6.3.4 Computational Complexity

In the previous section, we prove that obtaining a solution to both (6.8), (6.14) is simply a matter of solving the MSKCS Problem 2.3.10. In this section we prove that the decision version of the MSKCS Problem 2.3.11 is NP-complete. In the next section we offer a MILP formulation of the MSKCS problem.

Theorem 6.3.5 (NP-completeness)**.** *Problem 2.3.11 is NP-complete.*

Proof. In Chapter 5, we prove that the decision version of the MTSCS problem is NP-complete. Indeed, we prove that it is NP-hard, and offer a MILP formulation establishing that it is in NP. Now, we reduce the MTSCSP decision problem 2.3.9 to the MSKCS decision problem 2.3.11.

Given the tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W})$, a real number $t \geq 1$, and a natural number $K \geq 0$ that constitutes an instance of the MTSCS decision problem, we observe that Problem 2.3.11 on this instance is identical to the decision version of the MTSCSP by setting $t = T$. Thus, Problem 2.3.11 is NP-hard because the MTSCSP decision problem is. Observe that a potential solution $E = \bigcup_{o \in \mathcal{O}} E_o$ to Problem 2.3.11 can be verified as a solution to the MTSCS decision problem in time polynomial in $|L|$ by iterating over all vertices $j \in L$ to check $d^E(o, j) \leq t d^{\mathcal{B}}(o, j)$ for all $o \in \mathcal{O}$, and additionally checking that $\max_{o \in \mathcal{O}} |E_o| \leq K$. Thus Problem 2.3.11 is NP-hard. The same reduction can be used to reduce the MSKCS decision problem to the MTSCS decision problem (by setting $t = T$) and again, solutions to the latter can be verified as solutions to the former in time polynomial in $|L|$. Hence, Problem 2.3.11 is in NP and thus NP-complete. □

6.3.5 Computing an optimal control set: MILP Formulation

We now briefly summarize the proposed solution to problem (6.14). By Theorem 6.3.3, the solution is a tuple $(\bar{E}, \hat{\boldsymbol{w}})$, where $\hat{\boldsymbol{w}}$ can be approximated using (6.12), and $\bar{E}$ is the connection set of a MSKCS $E^{\hat{\boldsymbol{w}}}$. The MSKCS $E^{\hat{\boldsymbol{w}}}$ is computed for a lattice L on a configuration space $\mathcal{X}$ with start set $\mathcal{O}$ where costs of motions p between lattice vertices is given by $u(p, \hat{\boldsymbol{w}})$. These motions p are assumed to solve the motion planning problem 2.2.1 given cost $u(\cdot, \hat{\boldsymbol{w}})$. We now propose a MILP formulation of the MSKCS problem that can be used to compute $\bar{E}$. In particular, this MILP accommodates possible non-additive feature costs $\boldsymbol{f}$ provided Assumption 6.2.1 holds. Given $(\mathcal{X}, L, \mathcal{O}, \hat{\boldsymbol{w}})$ and a bound on the control

set size K,

$$\min\ T \tag{6.17a}$$
$$s.t.\ \forall o \in \mathcal{O} \tag{6.17b}$$
$$y_p^o \leq K, \qquad \forall p \in \mathcal{B}^{\hat{\boldsymbol{w}}} \tag{6.17c}$$
$$x_{ij}^{o'} - y_p^o \leq 0, \qquad \forall o' \in \mathcal{O}, \forall (i,j) \in S_p, \forall p \in \mathcal{B}^{\hat{\boldsymbol{w}}} \tag{6.17d}$$
$$f_l(P_{oi}) + f_l(p_{ij}) - f_l(P_{oj}) \leq M(1 - x_{ij}^o), \qquad \forall (i,j) \in L \times L', \forall \text{ additive } f_l \tag{6.17e}$$
$$f_l(P_{oi}) - f_l(P_{oj}) \leq M(1 - x_{ij}^o), \qquad \forall (i,j) \in L \times L', \forall \text{ maximizing } f_l \tag{6.17f}$$
$$f_l(p_{ij}) - f_l(P_{oj}) \leq M(1 - x_{ij}^o), \qquad \forall (i,j) \in L \times L', \forall \text{ maximizing } f_l \tag{6.17g}$$
$$\hat{\boldsymbol{w}}\boldsymbol{f}(P_{oj}) - z_j^o \leq N(1 - x_{ij}^o), \qquad \forall (i,j) \in L \times L' \tag{6.17h}$$
$$z_j^o \leq T c_{oj}, \qquad \forall j \in L' \tag{6.17i}$$
$$\sum_{i \in L} x_{ij}^o = 1, \qquad \forall j \in L' \tag{6.17j}$$
$$x_{ij}^o \in \{0,1\}, \qquad \forall (i,j) \in L \times L' \tag{6.17k}$$
$$y_p^o \in \{0,1\}, \qquad \forall p \in \mathcal{B}^{\hat{\boldsymbol{w}}}, \tag{6.17l}$$

where M, N are sufficiently large values. This MILP is very similar to (5.3) in Chapter 5. However, in (5.3) explicit values of M are given. These values depend on t, the acceptable $t-$error which is given as input to Problem 2.3.8 (the problem for which (5.3) is a MILP formulation). Unfortunately, the t-error T is a variable in (6.17), and therefore cannot be used to define M, N without making the problem non-linear. The other differences between (5.3), (6.17) are described here:

Optimization Function (6.17a) The optimization function (6.17a) and constraint (6.17i) seek to minimize the maximum $t-$error of the control set.

Constraint (6.17c) This constraint bounds the size of the control set. In particular, for each $o \in \mathcal{O}$, this constraint ensures that $|E_o| \leq K$.

Introduction of Path Feature Variables The MILP here introduces continuous variables $f_l(P_{oj})$ for each $j \in L', o \in \mathcal{O}$, and feature ϕ_l with feature function f_l. These variables represent the feature function f_l evaluated along the path P_{oj} from o to j in the Tree T^o described in Chapter 5. Constraints (6.17e), (6.17f), (6.17g), (6.17h) encode the cost continuity criteria given in (5.3e) in the MILP (5.3). In detail, if $x_{ij}^o = 1$ then (i,j) lies on a

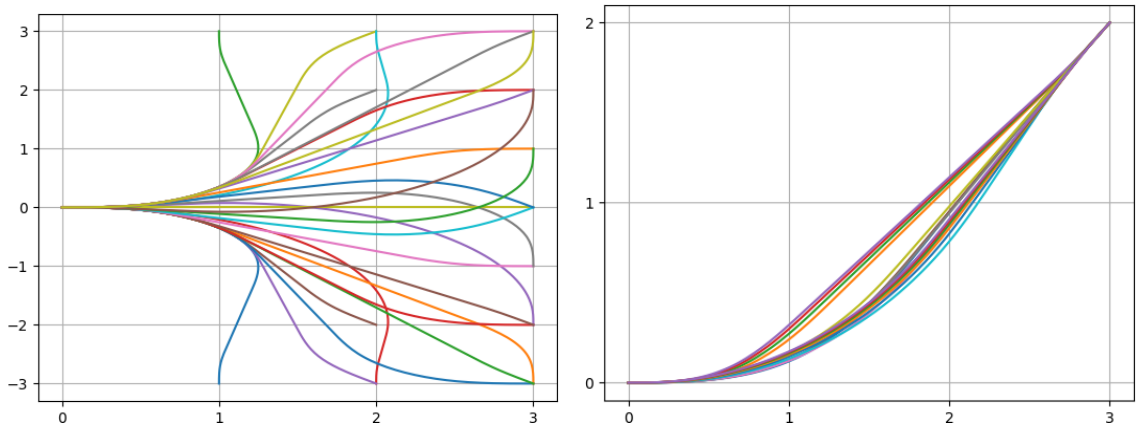
path from o to j in the tree T^o, and constraints (6.17e), (6.17f), (6.17g), (6.17h) reduce to

$$
\begin{aligned}
&f_l(P_{oj}) \geq f_l(P_{oi}) + f_l(P_{oj}),\ \forall \text{ additive features } f_l \\
&f_l(P_{oj}) \geq \max\left(f_l(P_{oi}), f_l(p_{ij})\right) \forall \text{ maximizing features } f_l. \\
&z_j^o \geq \hat{\boldsymbol{w}}\boldsymbol{f}(P_{oj}) = u(P_{oj}, \hat{\boldsymbol{w}}),
\end{aligned}
$$

which encode the cost continuity criteria. On the other hand, if $x_{ij}^o = 0$, then these constraints are trivially satisfied for sufficiently large M, N.

6.4 Evaluation

We now demonstrate the performance of the proposed approach in simulations. We consider a known, static environment, discretized into four-dimensional configurations. Each configuration of a (x, y) position, a heading θ and the current velocity v. We considered 3 discrete values of velocity, and 8 discrete headings (cardinal and ordinal). All used *environments* consisted of 15×15 grid of (x, y) positions and hence of $15 \times 15 \times 8 \times 3 = 5400$ configurations (vertices). The *lattice* had pre-computed motions for x-values between 0 and 4, y-values between -3 and 3, all 8 headings and all 3 speed values. Trajectories were computed using clothoids, a subset of the used lattice is illustrated in Figure 6.1. The start

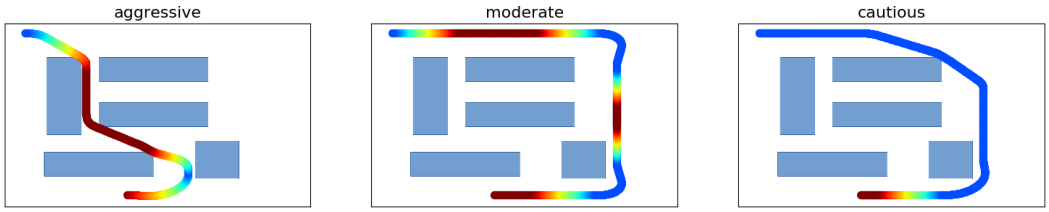


(a) Clothoid trajectory control set.

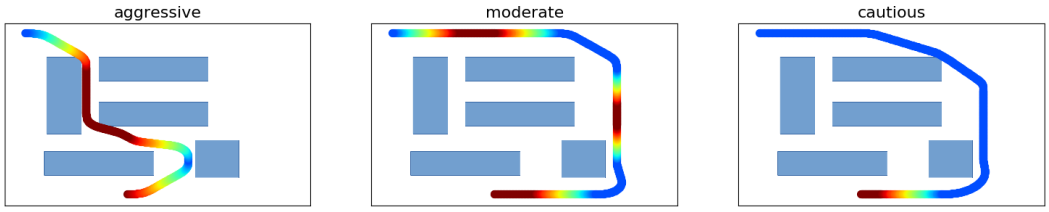
(b) Sample clothoid motions.

Figure 6.1: Examples of a lattice control set and different motions.

set was taken to have Property 2.3.1. In detail $\mathcal{O} = (0, 0, j, v_i), j \in \{0, \pi/4\}, i \in \{1, 2, 3\}$.



(a) Demonstrated behaviour



(b) Learned behaviour

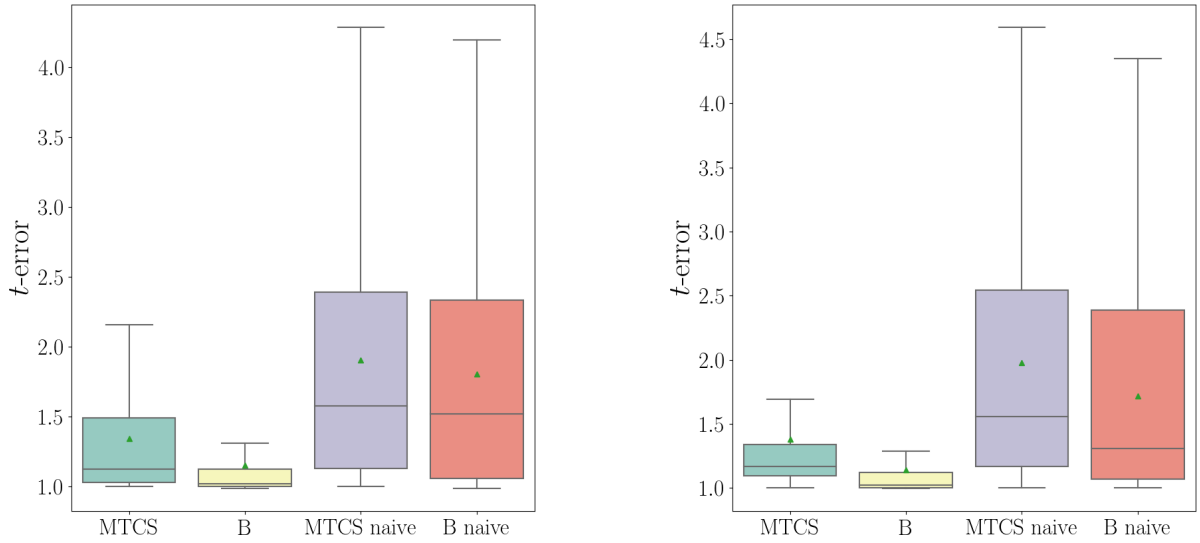
Figure 6.2: Demonstrated and learned behaviour in the training environment. The color of the path indicates speed, where blue corresponds to slow and red corresponds to fast.

Thus, the connection set $\bar{\mathcal{B}}$ for each start contains up to 672 connections. The workspace was considered to be $\mathbb{R}^2$.

We model users who evaluate motions based on three features: travel time, maximum longitudinal acceleration and maximum lateral acceleration. User demonstration were simulated by sampling features from the distribution in (6.10). We consider three preferences corresponding to different driving styles: An aggressive style which prefers short travel times, a cautious style that favours low accelerations, and a moderate style that balances the features more equally. For all user types we set the covariance in (6.10) to 0.1. We illustrate an example demonstration of each user in Figure 6.2a. Finally, the experiment comprises one training environment, shown in Figure 6.2 and two test environments.

6.4.1 Training Error

For each of the three driving style, we obtain three demonstrations for different start-goal pairs. We show that we can effectively learn user preferences despite this small number of demonstrations. We estimate $\hat{\boldsymbol{w}}$ for each user by computing the expectation in equation (6.12) using $N = 10$ samples and assuming a covariance of 0.05, i.e., we overestimate how accurately the user demonstration match their optimal motions. Finally, we run the experiment for minimal k-spanning connection sets of size 25, 50 and 100 over 40 trials each.



(a) Training Error (b) Test error

Figure 6.3: The t-error for all connections $\bar{\mathcal{B}}$ and the MSKCS connections $\bar{E}$ compared to a naive approach with $\bar{\mathcal{B}}_{\texttt{naive}}$ and $\bar{E}_{\texttt{naive}}$, respectively.

Figure 6.2 shows one of the training demonstrations starting at the bottom at high speed and going to the top left corner. We compare the optimal paths for the three different users, together with the paths that were planned using the learned control set $(\bar{E}, \hat{\boldsymbol{w}})$. While the shape of the paths is slightly different from the demonstrations, key characteristics of the user preferences are replicated. The aggressive user only breaks for sharp turns and immediately accelerates again. The moderate user avoids breaking too abruptly and thus cannot take the shortcut. Finally, the cautious user minimizes lateral and longitudinal acceleration and thus drives with minimal speed whenever possible. Figure 6.3a shows the t-error over all users and k values for the control set $(\bar{E}, \hat{\boldsymbol{w}})$ together with the error of the control set using all connections $(\bar{\mathcal{B}}, \hat{\boldsymbol{w}})$ in the lattice. We include a naive approach as a reference, where $\boldsymbol{w}_{\texttt{naive}}$ is sampled randomly and independent to the demonstrations. This serves as a baseline to illustrate the advantage of estimating driver behavior, i.e., the sensitivity of the cost function to user preferences $\boldsymbol{w}$. For the naive approach we show the error of a MSKCS $(\bar{E}_{\texttt{naive}}, \boldsymbol{w}_{\texttt{naive}})$ and of $(\bar{\mathcal{B}}_{\texttt{naive}}, \boldsymbol{w}_{\texttt{naive}})$. We observe that $(\bar{\mathcal{B}}, \hat{\boldsymbol{w}})$ achieves an average error of 1.16, with a median of 1.02. The MSKCS solution has a mean and median error of 1.34 and 1.13, respectively. Thus, the limited size of the control set leads to a mean cost that is 34% higher than the optimum, but for half of all cases the cost is at most 13% higher. In comparison, we observe that the naive approach yields a mean error

of 1.81 with a median of 1.52 when using all connections, and a slightly higher error for a MSKCS.

In Figure 6.4 we compare how different values for k influence the t-error and the planning time speedup when using a MSKCS. Generally, the t-error decreases as k increases, though we observe large differences between user types. For the cautious user the error does not exceed 1.1 on average, independent of k. While the aggressive user type achieves comparable values for $k = 100$, the error increases drastically for smaller k, reaching an average of ≈ 2.2 for $k = 25$ with a large deviation from the mean. The moderate user shows a more balanced result with the average not surpassing 1.5 for small k. Yet, the planning time speedup is less affected by the user type. For $k = 25$ we observe the highest speedup of ≈ 35 on the median for all three user types. For higher values of k the speedup decreases, but even using a connection set with $|\bar{E}| = 100$, the median path planning is at least 10 times faster than when using all connections $\mathcal{B}$. We conclude that using a

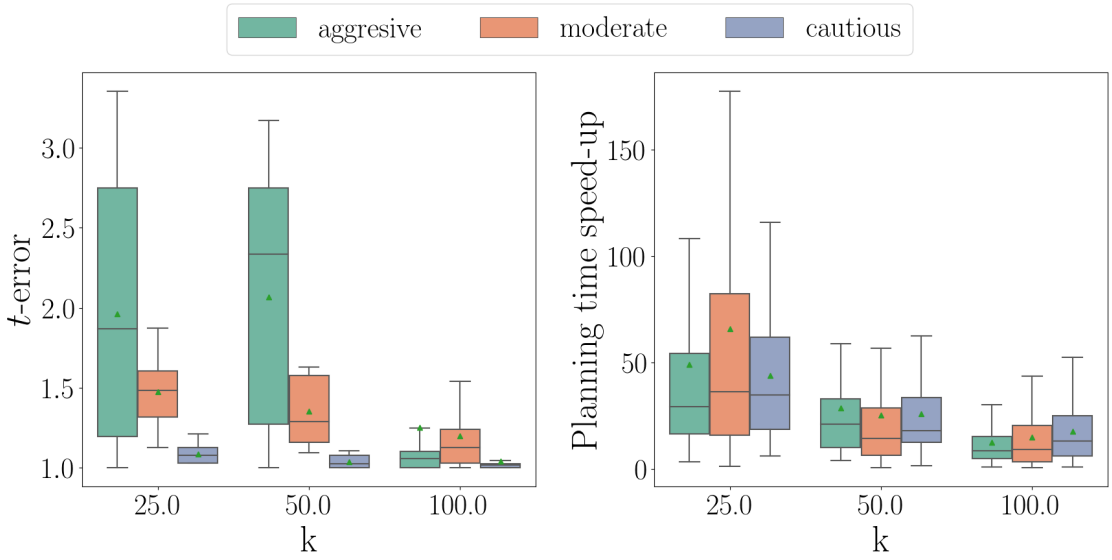


Figure 6.4: Training data: The t-error and planning time speedup for different sizes of the MSKCS and the different user types.

MSKCS drastically decreases planning time though the cost increases 34% compared to the optimum (twice the error of the best estimate). However, in half of all training examples the increase in cost is less than 10%. Between users, the performance benefit is similar, while the path quality differs.

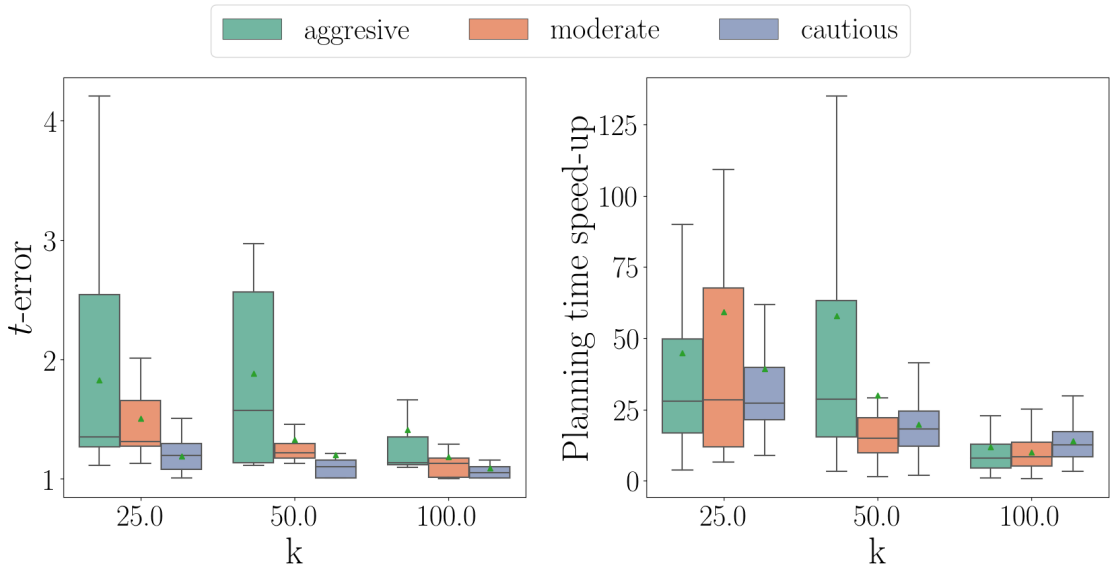


Figure 6.5: Test data: The t-error and planning time speedup for different sizes of the MSKCS and the different user types.

6.4.2 Test Error

The test error is evaluated on three different start–goal pairs for which no demonstrations were obtained in two different environments. In Figure 6.3b we show a similar analysis as we did for the training error. The error of the learned control set $(\bar{E}, \hat{\boldsymbol{w}})$ is slightly higher than in the training with a mean of 1.38 and a median of 1.17 while the deviation is smaller. The naive approach performs slightly better than in the training, with mean and median values for $(\bar{\mathcal{B}}_{\texttt{naive}}, \boldsymbol{w}_{\texttt{naive}})$ of 1.72 and 1.31, respectively. This indicates that the user behaviour might be less sensitive in the test scenarios. Figure 6.5 shows a comparable relationship between k and the t-error (left) and planning time speedup (right). The overall t-error is comparable in mean to the training, but increases with smaller k for the cautious user type. The aggressive user shows the highest error, with a mean of 1.8 for $k = 50$. The planning speedup shows a similar trend as the training data. However, the deviations differ more between user types. All three users yield median speedup factors of ≈ 30 for $k = 25$, but are still above 10 for $k = 100$.

We conclude that the learned control set $(\bar{E}, \hat{\boldsymbol{w}})$ achieves good t-errors in both training and test environments while allowing for a substantial planning time speedup. Increasing k, the t-error tends to decrease, approaching the error of the estimation. Even though the performance also decreases with growing k, we still obtain an improvement of more than a

factor of 10 for $k = 100$ compared to paths planned using the complete connection set $\bar{\mathcal{B}}$.

6.5 Discussion

We studied a novel approach for computing a control set that captures user preferences. First an estimate $\hat{\boldsymbol{w}}$ of the user weights, given data, and a set of motions using those weights are computed for all pairs $(s, j) \in \bar{\mathcal{B}}$. These motions are calculated to minimize a user cost function that is a linear combination of weights $\hat{\boldsymbol{w}}$ and trajectory features. Next, a set of connections $\bar{E} \subseteq \bar{\mathcal{B}}$ and its associated motions is determined. This control set minimizes the relative error of motions generated. We illustrate how both $\hat{\boldsymbol{w}}$ and $\bar{E}$ are computed, and validate our findings using a clothoid trajectory planner. The results illustrate that the control set given by $(\bar{E}, \hat{\boldsymbol{w}})$ is able to capture a user's preferences while greatly reducing online computation time relative to using the full connection set.

In future, this approach should be tested with real-world data, e.g., recorded driving behaviour. Further, these results should be extended to account for situational user weights.

Chapter 7

Discussion and Future Directions

This thesis was dedicated to investigating solutions and uses for the Minimum $t-$Spanning Control Set (MTSCS) problem. A $t-$spanning control set guarantees that each motion between lattice vertices can be approximated using motions in the control set to within a factor of t of their cost. Therefore, the problem of computing a smallest set with this guarantee is a problem that optimizes a trade-off between the size of the control set (i.e., the run time of an online graph-search) and the quality of the motions it can compute.

We began with a class of instances for which the MTSCS problem can be solved efficiently. In Chapter 3, we presented one such efficient algorithm to compute control sets for a square lattice in a configuration space whose configurations had only positional states and where the cost of a motion was given by the Euclidean distance between its endpoints. In this chapter, we showed that an intuitive solution to the continuous version of the problem persisted even when the plane was discretized into integer-valued lattice vertices. Further, bounds on both the size of the control set as well as the error between lattice-planned paths and free-space optimal paths were presented. Finally, this chapter illustrated that a MTSCS could achieve the same $t-$factor sub-optimality as a state of the art control set generation technique but with exponentially smaller control set size.

Chapter 3 served as motivation to consider more complex instances of the MTSCS problem. In particular, a class of higher-dimensional lattices where the cost of a motion was a user-specific quantity. The quality of a control set depends heavily on the quality of its motions. That is, to create a compound motion from motion primitives that is pleasing to a user, each motion primitive must itself be pleasing. This observation motivated the work presented in Chapter 4. Here, a method to compute trajectories that optimize a trade off between comfort and travel time was presented. The described technique involves

computing and iteratively refining a path and a velocity profile between two configurations. Though this work focused on comfort and travel time, other trajectory features could be incorporated into the proposed cost function without affecting the proposed technique.

Trajectories computed using the method proposed in Chapter 4 were used to validate the results of the next chapter. In Chapter 5, we proved that the MTSCS problem is NP-complete in general and we provided a MILP formulation of the problem. Combining the work presented in Chapters 4 and 5 results in method to compute a control set that 1) accommodates the preferences of a user, i.e., each motion in the control set is optimal for the user, 2) is of minimal size, and 3) can be used to generate compound motions that are within a factor of t from user-optimal. However, this work assumes that the preferences of a user are known and are available as a set of weights.

In Chapter 6, we illustrate that an intuitive procedure of learning weights, planning motions, and selecting a control set, is optimal when attempting to compute a control set that takes user preferences into account. That is, the problem of simultaneously estimating user preferences and selecting motion primitives can be optimally solved via a separation principle: first estimate the weights, then select a control set.

The inputs to the MTSCS problem is the tuple $(\mathcal{X}, L, \mathcal{O}, c, \mathcal{W})$ of configuration space $\mathcal{X}$, lattice L, starting set $\mathcal{O}$, cost c, and workspace $\mathcal{W}$, as well as a value $t \geq 1$. Given $\mathcal{X}, L, \mathcal{W}$, we have seen how a starting set $\mathcal{O}$ that generalizes the vertices of a lattice can be selected. We have also seen how costs can be estimated from demonstrations to inform the selection of a control set. Therefore, from $\mathcal{X}, L, \mathcal{W}$, and a user demonstrations, the remaining inputs to the MTSCS problem may be estimated given a value $t \geq 1$, and a control set can be computed. However, this work has not addressed critical questions in autonomous driving. In particular: how should L be computed? The lattices used in Chapters 5, 4 were selected by trial and error and according to what seemed reasonable given the circumstances. However, *how* a configuration space should discretized is surely as important a question as how to traverse the discretization. It is the opinion of the author that an iterative process of forming a lattice, selecting a control set, testing the control set, and then updating the lattice based on the efficacy of the control set during testing may result in better results than treating discretization and control set selection as two disjoint problems.

Situational specific lattices/user weights were also not addressed in this thesis. That is, we did not formally investigate how one could develop a set of lattices and user weights each tailored to a specific situations. This idea was used informally in Chapter 5 where one lattice was computed to traverse parking lots and another for highways. However, there are a myriad of other situations: city streets, merging ramps, emergency maneuvering, etc.

This begs the question: is there a way to compute a set of lattices for specific situations such that motions and vertices from one can be smoothly concatenated to produce vertices of another? Perhaps via a third transitional lattice? What is the smallest set of lattices required to ensure that every maneuver in every situation is perform-able within some specified error bound, and that there is a sequence of motions taking any vertex in any of these lattices to any other lattices' vertices? It is the opinion of this author that by treating situationally-derived lattices like configurations in a *meta*-lattice, one may be able to employ similar $t-$spanning techniques as those derived in this work to solve this last problem.

In addition to lattice selection, this work does not address trajectory tracking or vehicle control, and we have not tested our findings in real-world situations. While available literature suggests that smoother trajectories require less control effort and result in smaller tracking error (a suggestion modestly verified in Chapter 4), it is left for future work to verify that this holds in practice for techniques proposed in this thesis.

References

[1] Pieter Abbeel, Dmitri Dolgov, Andrew Y Ng, and Sebastian Thrun. Apprenticeship learning for motion planning with application to parking lot navigation. In *2008 IEEE/RSJ IROS*, pages 1083–1090. IEEE, 2008.

[2] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[3] Olov Andersson, Oskar Ljungqvist, Mattias Tiger, Daniel Axehill, and Fredrik Heintz. Receding-horizon lattice-based motion planning with dynamic obstacle avoidance. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 4467–4474, 2018.

[4] Oktay Arslan, Karl Berntorp, and Panagiotis Tsiotras. Sampling-based algorithms for optimal motion planning using closed-loop prediction. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4991–4996. IEEE, 2017.

[5] Il Bae, Jaeyoung Moon, and Jeongseok Seo. Toward a comfortable driving experience for a self-driving shuttle bus. *Electronics*, 8(9):943, 2019.

[6] Holger Banzhaf, Nijanthan Berinpanathan, Dennis Nienhüser, and J Marius Zöllner. From g2 to g3 continuity: Continuous curvature rate steering functions for sampling-based nonholonomic motion planning. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 326–333. IEEE, 2018.

[7] B.A. Barsky and T.D. DeRose. Geometric continuity of parametric curves: three equivalent characterizations. *IEEE Computer Graphics and Applications*, 9(6):60–69, 1989.

[8] Kristoffer Bergman, Oskar Ljungqvist, and Daniel Axehill. Improved optimization of motion primitives for motion planning in state lattices. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2307–2314, 2019.

[9] Kristoffer Bergman, Oskar Ljungqvist, Torkel Glad, and Daniel Axehill. An optimization-based receding horizon trajectory planning algorithm. *IFAC-PapersOnLine*, 53(2):15550–15557, 2020.

[10] C Guarino Lo Bianco and Aurelio Piazzi. Optimal trajectory planning with quintic g2-splines. In *2000 IEEE Intelligent Vehicles Symposium (IV)*, pages 620–625, 2000.

[11] Lorenz T Biegler. Solution of dynamic optimization problems by successive quadratic programming and orthogonal collocation. *Computers & chemical engineering*, 8(3-4):243–247, 1984.

[12] Alexander Botros and Stephen L. Smith. Computing a minimal set of t-spanning motion primitives for lattice planners. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2328–2335, 2019.

[13] Alexander Botros, Nils Wilde, and Stephen L Smith. Learning control sets for lattice planners from user preferences. In *The 14th International Workshop on the Algorithmic Foundations of Robotics (WAFR), volume: To appear*, 2020.

[14] Jingwei Chen, Robert C Holte, Sandra Zilles, and Nathan R Sturtevant. Front-to-end bidirectional heuristic search with near-optimal node expansions. *arXiv preprint arXiv:1703.03868*, 2017.

[15] Ernest Cheung, Aniket Bera, Emily Kubin, Kurt Gray, and Dinesh Manocha. Identifying driver behaviors using trajectory features for vehicle navigation. In *2018 IEEE/RSJ IROS*, pages 3445–3452. IEEE, 2018.

[16] Howie M Choset, Kevin M Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, Sebastian Thrun, and Ronald C Arkin. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[17] Laurene Claussmann, Marc Revilloud, Dominique Gruyer, and Sébastien Glaser. A review of motion planning for highway autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 21(5):1826–1848, 2019.

[18] Charles M Close, Dean K Frederick, and Jonathan C Newell. *Modeling and analysis of dynamic systems*. John Wiley & Sons, 2001.

[19] Benjamin J Cohen, Sachin Chitta, and Maxim Likhachev. Search-based planning for manipulation with motion primitives. In *2010 IEEE ICRA*, pages 2902–2908. IEEE, 2010.

[20] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[21] Kenny Daniel, Alex Nash, Sven Koenig, and Ariel Felner. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, 39:533–579, 2010.

[22] Ryan De Iaco, Stephen L Smith, and Krzysztof Czarnecki. Learning a lattice planner control set for autonomous vehicles. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 549–556. IEEE, 2019.

[23] Jacques Desrosiers, Yvan Dumas, Marius M Solomon, and François Soumis. Time constrained routing and scheduling. *Handbooks in operations research and management science*, 8:35–139, 1995.

[24] Mihir Dharmadhikari, Tung Dang, Lukas Solanka, Johannes Loje, Huan Nguyen, Nikhil Khedekar, and Kostas Alexis. Motion primitives-based path planning for fast and agile exploration using aerial robots. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 179–185, 2020.

[25] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. *Ann Arbor*, 1001(48105):18–80, 2008.

[26] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010.

[27] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497, 1957.

[28] Yi Fang, Jie Hu, Wenhai Liu, Quanquan Shao, Jin Qi, and Yinghong Peng. Smooth and time-optimal s-curve trajectory planning for automated robots and machines. *Mechanism and Machine Theory*, 137:127–153, 2019.

[29] T. Fraichard and A. Scheuer. From reeds and shepp's to continuous-curvature paths. *IEEE Transactions on Robotics*, 20(6):1025–1035, Dec 2004.

[30] Hiroshi Fuji, Jingyu Xiang, Yuichi Tazaki, Blaine Levedahl, and Tatsuya Suzuki. Trajectory planning for automated parking using multi-resolution state roadmap considering non-holonomic constraints. In *2014 IEEE Intelligent Vehicles Symposium (IV)*, pages 407–413. IEEE, 2014.

[31] Jonathan D Gammell, Timothy D Barfoot, and Siddhartha S Srinivasa. Informed asymptotically optimal anytime search. *arXiv preprint arXiv:1707.01888*, 2017.

[32] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 3067–3074. IEEE, 2015.

[33] Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

[34] David González, Joshué Pérez, Vicente Milanés, and Fawzi Nashashibi. A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 17(4):1135–1145, 2015.

[35] David Sierra González, Ozgur Erkent, Víctor Romero-Cano, Jilles Dibangoye, and Christian Laugier. Modeling driver behavior from demonstrations in dynamic environments using spatiotemporal lattices. In *2018 IEEE ICRA*, pages 1–7. IEEE, 2018.

[36] Michael Grant and Stephen Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. http://stanford.edu/~boyd/graph_dcp.html.

[37] Tianyu Gu, Jason Atwood, Chiyu Dong, John M Dolan, and Jin-Woo Lee. Tunable and stable real-time trajectory planning for urban autonomous driving. In *2015 IEEE/RSJ IROS*, pages 250–256. IEEE, 2015.

[38] Shilpa Gulati, Chetan Jhurani, and Benjamin Kuipers. A nonlinear constrained optimization framework for comfortable and customizable motion planning of nonholonomic mobile robots-part i. *arXiv preprint arXiv:1305.5024*, 2013.

[39] Hongyan Guo, Chen Shen, Hui Zhang, Hong Chen, and Rui Jia. Simultaneous trajectory planning and tracking using an mpc method for cyber-physical systems:

A case study of obstacle avoidance for an intelligent vehicle. *IEEE Transactions on Industrial Informatics*, 14(9):4273–4283, 2018.

[40] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2019.

[41] L. Han, H. Yashiro, H. Tehrani Nik Nejad, Q. H. Do, and S. Mita. Bezier curve based path planning for autonomous vehicle in urban environment. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1036–1042, June 2010.

[42] Peter A Hancock, Illah Nourbakhsh, and Jack Stewart. On the future of transportation in an era of automated and autonomous vehicles. *Proceedings of the National Academy of Sciences*, 116(16):7684–7691, 2019.

[43] Daniel Damir Harabor and Alban Grastien. An optimal any-angle pathfinding algorithm. In *Twenty-Third International Conference on Automated Planning and Scheduling*, 2013.

[44] Daniel Damir Harabor, Alban Grastien, Dindar Öz, and Vural Aksakalli. Optimal any-angle pathfinding in practice. *Journal of Artificial Intelligence Research*, 56:89–118, 2016.

[45] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[46] R. Haschke, E. Weitnauer, and H. Ritter. On-line planning of time-optimal, jerk-limited trajectories. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3248–3253, 2008.

[47] Gabriel M Hoffmann, Claire J Tomlin, Michael Montemerlo, and Sebastian Thrun. Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In *American Control Conference (ACC)*, pages 2296–2301, 2007.

[48] Neville Hogan and Dagmar Sternad. Sensitivity of smoothness measures to movement duration, amplitude, and arrests. *Journal of motor behavior*, 41(6):529–534, 2009.

[49] Quanan Huang and Huiyi Wang. Fundamental study of jerk: evaluation of shift quality and ride comfort. Technical report, SAE Technical Paper, 2004.

[50] Lucas Janson, Brian Ichter, and Marco Pavone. Deterministic sampling-based motion planning: Optimality, complexity, and performance. *The International Journal of Robotics Research*, 37(1):46–61, 2018.

[51] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, 34(7):883–921, 2015.

[52] Laura Jarin-Lipschitz, James Paulos, Raymond Bjorkman, and Vijay Kumar. Dispersion-minimizing motion primitives for search-based motion planning. *arXiv preprint arXiv:2103.14603*, 2021.

[53] Morton I Kamien and Nancy Lou Schwartz. *Dynamic optimization: the calculus of variations and optimal control in economics and management*. Dover Publications, 2 edition, 2013.

[54] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[55] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[56] Adam Kasperski and Pawel Zielinski. An approximation algorithm for interval data minmax regret combinatorial optimization problems. *Inf. Process. Lett.*, 97(5):177–180, 2006.

[57] Christos Katrakazas, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60:416–442, 2015.

[58] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[59] Alonzo Kelly and Bryan Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *The International Journal of Robotics Research*, 22(8):583 – 601, July 2003.

[60] Sebastian Klemm, Jan Oberländer, Andreas Hermann, Arne Roennau, Thomas Schamm, J Marius Zollner, and Rüdiger Dillmann. Rrt-connect: Faster, asymptotically optimal motion planning. In *2015 IEEE international conference on robotics and biomimetics (ROBIO)*, pages 1670–1677. IEEE, 2015.

[61] Bernhard Korte and Jens Vygen. *Combinatorial optimization*. Springer, 6 edition, 2018.

[62] Bernhard Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*. Springer, 6 edition, 2018.

[63] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[64] Kostas J Kyriakopoulos and George N Saridis. Minimum jerk for trajectory planning and control. *Robotica*, 12(2):109–113, 1994.

[65] Jae-Yeong Lee and Wonpil Yu. A coarse-to-fine approach for fast path finding for mobile robots. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5414–5419. IEEE, 2009.

[66] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 163–168. IEEE, 2011.

[67] Xiaohui Li, Zhenping Sun, Dongpu Cao, Zhen He, and Qi Zhu. Real-time trajectory planning for autonomous urban driving: Framework, algorithms, and verifications. *IEEE/ASME Transactions on Mechatronics*, 21(2):740–753, 2015.

[68] Daniel Liberzon. *Calculus of variations and optimal control theory*. Princeton university press, 2011. Available at http://http://liberzon.csl.illinois.edu/teaching/cvoc.pdf.

[69] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. *Advances in neural information processing systems*, 16:767–774, 2003.

[70] Yucong Lin and Srikanth Saripalli. Sampling-based path planning for uav collision avoidance. *IEEE Transactions on Intelligent Transportation Systems*, 18(11):3179–3192, 2017.

[71] Thomas Lipp and Stephen Boyd. Minimum-time speed optimisation over a fixed path. *International Journal of Control*, 87(6):1297–1311, 2014.

[72] Changliu Liu, Yizhou Wang, and Masayoshi Tomizuka. Boundary layer heuristic for search-based nonholonomic path planning in maze-like environments. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 831–836. IEEE, 2017.

[73] Matthew McNaughton, Chris Urmson, John M Dolan, and Jin-Woo Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4889–4895, 2011.

[74] Bernard Michini, Thomas J Walsh, Ali-Akbar Agha-Mohammadi, and Jonathan P How. Bayesian nonparametric reward learning from demonstration. *IEEE Transactions on Robotics*, 31(2):369–386, 2015.

[75] Alex Nash and Sven Koenig. Any-angle path planning. *AI Magazine*, 34(4):85–107, 2013.

[76] Rui Oliveira, Mårcello Cirillo, Bo Wahlberg, et al. Combining lattice-based planning and path optimization in autonomous heavy duty vehicle applications. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 2090–2097. IEEE, 2018.

[77] Rui Oliveira, Pedro F Lima, Marcello Cirillo, Jonas Mårtensson, and Bo Wahlberg. Trajectory generation using sharpness continuous dubins-like paths with applications in control of heavy-duty vehicles. In *2018 European Control Conference (ECC)*, pages 935–940. IEEE, 2018.

[78] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[79] Malayandi Palan, Nicholas C Landolfi, Gleb Shevchuk, and Dorsa Sadigh. Learning reward functions by integrating human demonstrations and preferences. *arXiv preprint arXiv:1906.08928*, 2019.

[80] Luigi Palmieri, Leonard Bruns, Michael Meurer, and Kai O Arras. Dispertio: Optimal sampling for safe deterministic motion planning. *IEEE Robotics and Automation Letters*, 5(2):362–368, 2019.

[81] Aleksandr I Panov, Konstantin S Yakovlev, and Roman Suvorov. Grid path planning with deep reinforcement learning: Preliminary results. *Procedia computer science*, 123:347–353, 2018.

[82] Aditya A Paranjape, Kevin C Meier, Xichen Shi, Soon-Jo Chung, and Seth Hutchinson. Motion primitives and 3d path planning for fast flight through a forest. *The International Journal of Robotics Research*, 34(3):357–377, 2015.

[83] Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. Using probabilistic movement primitives in robotics. *Autonomous Robots*, 42(3):529–551, 2018.

[84] Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. Probabilistic movement primitives. *Advances in neural information processing systems*, 26:2616–2624, 2013.

[85] David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989.

[86] Scott Drew Pendleton, Wei Liu, Hans Andersen, You Hong Eng, Emilio Frazzoli, Daniela Rus, and Marcelo H Ang. Numerical approach to reachability-guided sampling-based motion planning under differential constraints. *IEEE Robotics and Automation Letters*, 2(3):1232–1239, 2017.

[87] A. Piazzi and C. Guarino Lo Bianco. Quintic g/sup 2/-splines for trajectory planning of autonomous vehicles. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 198–203, Oct 2000.

[88] Mihail Pivtoraiko and Alonzo Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3231–3237. IEEE, 2005.

[89] Mihail Pivtoraiko and Alonzo Kelly. Kinodynamic motion planning with state lattice motion primitives. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2172–2179. IEEE, 2011.

[90] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[91] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4):193–204, 1970.

[92] M. Raineri and C. Guarino Lo Bianco. Jerk limited planner for real-time applications requiring variable velocity bounds. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 1611–1617, 2019.

[93] Abhijeet Ravankar, Ankit A Ravankar, Yukinori Kobayashi, Yohei Hoshino, and Chao-Chung Peng. Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges. *Sensors*, 18(9):3170, 2018.

[94] Nicolás Rivera, Carlos Hernández, Nicolás Hormazábal, and Jorge A Baier. The 2 to the k neighborhoods for grid path planning. *Journal of Artificial Intelligence Research*, 67:81–113, 2020.

[95] Jin Woo Ro, Partha S. Roop, and Avinash Malik. A new safety distance calculation for rear-end collision avoidance. *IEEE Transactions on Intelligent Transportation Systems*, 22(3):1742–1747, 2021.

[96] BW Robinson, L Rodegerdts, Wade Scarborough, W Kittelson, R Troutbeck, Werner Brilon, Lothar Bondzio, Ken Courage, Michael Kyte, John Mason, et al. Roundabouts: An informational guide. federal highway administration. *Turner-Fairbank Highway Research Center*, 3, 2000.

[97] Martin Rufli and Roland Siegwart. On the design of deformable input-/state-lattice graphs. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3071–3077. IEEE, 2010.

[98] Alexandru Rusu, Sabine Moreno, Yoko Watanabe, Mathieu Rognant, and Michel Devy. State lattice generation and nonholonomic path planning for a planetary exploration rover. In *65th International Astronautical Congress 2014 (IAC 2014)*, volume 2, page 953, 2014.

[99] Dorsa Sadigh, Anca D Dragan, Shankar Sastry, and Sanjit A Seshia. Active preference-based learning of reward functions. In *Robotics: Science and Systems (RSS)*, 2017.

[100] Edward Schmerling, Lucas Janson, and Marco Pavone. Optimal sampling-based motion planning under differential constraints: the driftless case. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2368–2375. IEEE, 2015.

[101] Steven H Schot. Jerk: the time rate of change of acceleration. *American Journal of Physics*, 46(11):1090–1094, 1978.

[102] Dong Hun Shin, Sanjiv Singh, and W Whittaker. Path generation for a robot vehicle using composite clothoid segments. *IFAC Proceedings Volumes*, 25(6):443–448, 1992.

[103] J. A. R. Silva and V. Grassi. Clothoid-based global path planning for autonomous vehicles in urban scenarios. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4312–4318, May 2018.

[104] Kiril Solovey, Lucas Janson, Edward Schmerling, Emilio Frazzoli, and Marco Pavone. Revisiting the asymptotic optimality of rrt. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2189–2195. IEEE, 2020.

[105] Wataru Takeda. The exact order of the number of lattice points visible from the origin. *arXiv preprint arXiv:1608.02703*, 2016.

[106] Mattias Tiger, David Bergström, Andreas Norrstig, and Fredrik Heintz. Enhancing lattice-based motion planning with introspective learning and reasoning. *IEEE Robotics and Automation Letters*, 6(3):4385–4392, 2021.

[107] Tansel Uras and Sven Koenig. Fast near-optimal path planning on state lattices with subgoal graphs. In *Eleventh Annual Symposium on Combinatorial Search*, 2018.

[108] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[109] Diederik Verscheure, Bram Demeulenaere, Jan Swevers, Joris De Schutter, and Moritz Diehl. Time-optimal path tracking for robots: A convex optimization approach. *IEEE Transactions on Automatic Control*, 54(10):2318–2327, 2009.

[110] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[111] Larry Wasserman. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media, 2013.

[112] Wenda Xu, Junqing Wei, J. M. Dolan, Huijing Zhao, and Hongbin Zha. A real-time motion planner with trajectory optimization for autonomous vehicles. In *2012 IEEE International Conference on Robotics and Automation*, pages 2061–2067, May 2012.

[113] Moritz Werling, Julius Ziegler, Sören Kammel, and Sebastian Thrun. Optimal trajectory generation for dynamic street scenarios in a frenet frame. In *2010 IEEE International Conference on Robotics and Automation*, pages 987–993. IEEE, 2010.

[114] Taeg-Keun Whangbo. Efficient modified bidirectional a* algorithm for optimal route-finding. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 344–353. Springer, 2007.

[115] Edmund Taylor Whittaker. *A treatise on the analytical dynamics of particles and rigid bodies*. CUP Archive, 1937.

[116] Kwangjin Yang, Sangwoo Moon, Seunghoon Yoo, Jaehyeon Kang, Nakju Lett Doh, Hong Bong Kim, and Sanghyun Joo. Spline-based rrt path planner for non-holonomic robots. *Journal of Intelligent & Robotic Systems*, 73(1):763–782, 2014.

[117] Kwangjin Yang and Salah Sukkarieh. An analytical continuous-curvature path-smoothing algorithm. *IEEE Transactions on Robotics*, 26(3):561–568, 2010.

[118] Xiaojing Zhang, Alexander Liniger, and Francesco Borrelli. Optimization-based collision avoidance. *IEEE Transactions on Control Systems Technology*, 29(3):972–983, 2021.

[119] Yu Zhang, Huiyan Chen, Steven L Waslander, Jianwei Gong, Guangming Xiong, Tian Yang, and Kai Liu. Hybrid trajectory planning for autonomous driving in highly constrained environments. *IEEE Access*, 6:32800–32819, 2018.

[120] Zhijie Zhu, Edward Schmerling, and Marco Pavone. A convex optimization approach to smooth trajectories for motion planning with car-like robots. In *2015 IEEE Conference on Decision and Control (CDC)*, pages 835–842. IEEE, 2015.

[121] Julius Ziegler, Philipp Bender, Thao Dang, and Christoph Stiller. Trajectory planning for bertha—a local, continuous method. In *2014 IEEE intelligent vehicles symposium proceedings*, pages 450–457. IEEE, 2014.

[122] Julius Ziegler, Philipp Bender, Markus Schreiber, Henning Lategahn, Tobias Strauss, Christoph Stiller, Thao Dang, Uwe Franke, Nils Appenrodt, Christoph G Keller, et al. Making bertha drive—an autonomous journey on a historic route. *IEEE Intelligent transportation systems magazine*, 6(2):8–20, 2014.

APPENDICES

Appendix A

Proofs of Results in Chapter 3

A.1 Lemma 3.3.6

Lemma A.1.1 (BEW Paths Using E)**.** *Let $t \geq 1$, and $w = W[(x_1, y_1), (x_2, y_2)]$ be a BEW that is bounded by a control set E for a lattice L^n. For any $v \in w$ if $k_1, k_2 \in \mathbb{N}$ such that $v = (k_1 x_1 + k_2 x_2, k_1 y_1 + k_2 y_2)$, then*

$$d^E(v) = k_1 ||x_1, y_1|| + k_2 ||x_2, y_2||. \tag{A.1}$$

Further, if E^* is any control set, then for all $v \in w$,

$$\left(d^{E^*}(v) < d^E(v)\right) \implies \left(\exists u \in E^* \cap w^{\mathrm{o}}\right). \tag{A.2}$$

Proof. Let $(x_1, y_1) = (l_1, \theta_1), (x_2, y_2) = (l_2, \theta_2)$, then by definition of a wedge, $\theta_1 < \theta_2$. The second result (A.2) follows from the first result (A.1). Indeed, suppose that $E^* \cap w^{\mathrm{o}} = \emptyset$, and let $\bar{E} = E^* \cup \{(x_1, y_1), (x_2, y_2)\}$. Then $d^{E^*}(v) \geq d^{\bar{E}}(v)$ because $E^* \subseteq \bar{E}$ and each relative motion in E^* is available for paths using $\bar{E}$. Observe that $\bar{E} \cap w^{\mathrm{o}} = \emptyset, (x_1, y_1), (x_2, y_2) \in \bar{E}$ implies that w is bounded by $\bar{E}$. Therefore, $d^{\bar{E}}(v) = k_1 l + 1 + k_2 l_2 = d^E(v)$ (from (A.1)) implying that $d^{E^*}(v) \geq d^{\bar{E}}(v) = d^E(v)$ which establishes the contrapositive of (A.2).

Therefore, it suffices to prove (A.1). Let $L = k_1 l_1 + k_2 l_2$. Because w is bounded by E, it must hold that $(x_1, y_1), (x_2, y_2) \in E$ thus $d^E(v) \leq L$. Therefore, it suffices to show that $d^E(v) \geq L$. We offer a proof of this by way of contradiction. Suppose $d^E(v) < L$, and let $v = (x, y)$. Then there must exist a relative motion $(x_3, y_3) \in E$ in a path using E to v implying that $d^E(v) = ||x_3, y_3|| + d^E(x - x_3, y - y_3) \geq ||u|| + ||x - x_3, y - y_3||$ by

the triangle inequality. We establish the result in the worst case: $(x - x_3, y - y_3) \in E$ implying that $d^E(x - x_3, y - y_3) = ||x - x_3, y - y_3||$, and $d^E(v) = ||u|| + ||x - x_3, y - y_3||$. Let $(x_3, y_3) = (l_3, \theta_3), (x - x_3, y - y_3) = (l_4, \theta_4)$ where, without loss of generality, $\theta_3 < \theta_4$. Because w is bounded by E – implying that $E \cap w^o = \emptyset$ – we may conclude that $(x_3, y_3), (x - x_3, y - y_3) \notin w^o$ thus $\theta_3 < \theta_1, \theta_4 > \theta_2$. Therefore, the vertex (x_3, y_3) cannot appear to the left of the line passing through (k_1x_1, k_1, y_1). Indeed, if this were not the case, then $\theta_4 \in [\theta_1, \theta_2]$ which cannot happen if $(y - y_3, x - x_3) \notin w$. This is illustrated in Figure A.1 (Left). Therefore, (x_3, y_3) must appear to the right of the line passing through (k_1x_1, k_1y_1) and (x, y) as illustrated in Figure A.1 (Right). Let A, B, C, D be the lengths

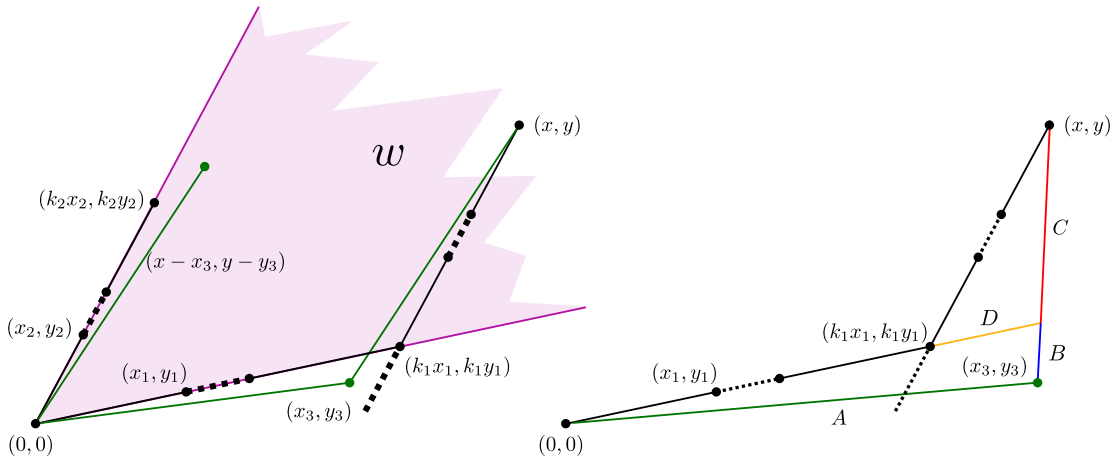


Figure A.1: (Left) Graphical proof that (x_3, y_3) cannot appear to the left of the line containing (x, y) and $k_1(x_1, y_1)$. (Right) Graphical proof that the shortest distance to (x, y) in E does not contain (x_3, y_3)

of the green, blue, red, and orange lines, respectively, as illustrated in Figure A.1 (Right). Then, observe by repeated applications of the triangle inequality:

$$L = k_1l_1 + k_2l_2 \leq k_1l_1 + D + C \leq A + B + C = l_3 + l_4 = d^E(x, y).$$

The second inequality comes from the observation that the orange line is colinear to the vector (k_1x_1, k_1y_1). If the above holds with equalities, then (A.1) is established. Otherwise, a contradiction is reached using the definition of $d^E(x, y)$. □

A.2 Lemma 3.3.7

Lemma A.2.1. *Let w be a BEW on a lattice L^n that is bounded by a control set E. Let u, v be two vertices in w. Then $m_w(u) \geq m_w(v)$ if and only if u lies above the line passing through v and the origin.*

Proof. Let $w = W[(x_1, y_1), (x_2, y_2)]$. Because w is a BEW, there must exist natural numbers k_1, k_2, q_1, q_2 such that $u = k_1(x_1, y_1) + k_2(x_2, y_2), v = q_1(x_1, y_1) + q_2(x_2, y_2)$. Let $l_1 = ||x_1, y_1||, l_2 = ||x_2, y_2||$. Then

$$m_w(u) \geq m_w(v) \iff \frac{k_2 l_2}{k_1 l_1} \geq \frac{q_2 l_2}{q_1 l_1} \iff \frac{k_2}{k_1} \geq \frac{q_2}{q_1} \iff k_2 q_1 \geq k_1 q_2 \tag{A.3}$$

Letting Δ_u, Δ_v denote the slope of the lines passing through the origin and u, v, respectively, we observe by (A.3) that

$$\begin{aligned} \Delta_u \geq \Delta_v &\iff \frac{k_2 y_1 + k_1 y_1}{k_2 x_2 + k_1 x_1} \geq \frac{q_2 y_1 + q_1 y_1}{q_2 x_2 + q_1 x_1} \\ &\iff 0 \leq (x_1 y_2 - x_2 y_1)(k_2 q_1 - k_1 q_2) \end{aligned}$$

Finally, observe that it is assumed in the definition of a BEW, that the angle formed between the $x-$axis and the line passing through the origin and (x_2, y_2) is larger than that for (x_1, y_1) implying that $y_2/x_2 > y_1/x_1$. Therefore, $x_1 y_2 - x_2 y_1 > 0$, and we conclude that $\Delta_u \geq \Delta_v$ if and only if (A.3) holds. □

A.3 Lemma 3.3.8

Lemma A.3.1 ($t-$Error for BEWs)**.** *Let $t \geq 1$, and consider lattice L^n with control set E and BEW $w = W[(x_1, y_1), (x_2, y_2)]$. If w is bounded by E, then the $t-$error of any point $(x, y) \in w^o$ is given by*

$$t^E(x, y) = \frac{1 + m_w(x, y)}{\sqrt{1 + m_w(x, y)^2 + 2m_w(x, y)B_w}}. \tag{A.4}$$

If, on the other hand, $(x_1, y_1) \in E$, and $(x_2, y_2) \notin E$ is $t-$reachable using E, then

$$t^E(x, y) \leq \frac{1 + t m_w(x, y)}{\sqrt{1 + m_w(x, y)^2 + 2m_w(x, y)B_w}}, \ \forall (x, y) \in w^o. \tag{A.5}$$

Finally, if $(x_2, y_2) \in E$ *and* $(x_1, y_1) \notin E$ *is* $t-$*reachable using* E*, then*

$$t^E(x,y) \leq \frac{t + m_w(x,y)}{\sqrt{1 + m_w(x,y)^2 + 2m_w(x,y)B_w}}, \ \forall (x,y) \in w^{\circ}. \tag{A.6}$$

Proof. Let $l_1 = ||x_1, y_1||, l_2 = ||x_2, y_2||$. If w is a BEW that is bounded by E, then by definition of an interior vertex of a BEW, there must exist $k_1, k_2 \in \mathbb{N} - \{0\}$ such that $(x,y) = k_1(x_1,y_1) + k_2(x_2,y_2)$. By Lemma 3.3.6, it must hold that $d^E(x,y) = k_1 l_1 + k_2 l_2$. Therefore, by the definition of the $t-$error and the law of cosines,

$$t^E(x,y) = \frac{k_1 l_1 + k_2 l_2}{\sqrt{k_1^2 l_1^2 + k_2^2 l_2^2 + 2k_1 k_2 l_1 l_2 B_w}}. \tag{A.7}$$

Dividing the numerator and denominator of (A.7) by $k_1 l_1$, and replacing $m_w(x,y) = k_2 l_2/(k_1 l_2)$ in the result yields (A.4).

If $(x_1, y_1) \in E$, and (x_2, y_2) is $t-$reachable using E, then $d^E(x_1, y_1) = l_1$ and $d^E(x_2, y_2) \leq t l_2$. Replacing this and the definition of m in (A.7) yields (A.5). Finally, if $(x_2, y_2) \in E$ and (x_1, y_1) is $t-$reachable using E, then $d^E(x_2, y_2) = l_2, d^E(x_1, y_1) \leq t l_1$. Replacing this the definition of m in (A.7) yields (A.6). □

A.4 Lemma 3.3.9

Lemma A.4.1 (t-Reachable Vertices in a BEW using E)**.** *Let* $t > 1$ *and consider a lattice* L^n *with control set* $E \subseteq L^n$ *and BEW* $w = W[(x_1, y_1), (x_2, y_2)]$ *that is bounded by* E*. Then the following are true:*

1. *If* $B_w \geq (2t^{-2} - 1)$*, then every vertex in* w *is* $t-$*reachable using* E*. If* $n = \infty$*, then this requirement is necessary as well as sufficient.*

2. *If not every vertex in* w *is* $t-$*reachable using* E*, then the set of all non* $t-$*reachable points* (x,y) *in* w *using* E *is exactly equal to*

$$\begin{aligned} \bar{w} =& \{(x,y) \in w \ : \ m_w(x,y) \in (\underline{m}_w, \overline{m}_{cirt})\}, \ \textit{where} \\ \underline{m}_w =& \frac{t\left(t(1-B_w) - \sqrt{(B_w - 1)(B_w t^2 + t^2 - 2)}\right)}{t^2 - 1} - 1, \\ \overline{m}_w =& \frac{t\left(t(1-B_w) + \sqrt{(B_w - 1)(B_w t^2 + t^2 - 2)}\right)}{t^2 - 1} - 1. \end{aligned} \tag{A.8}$$

Proof. We begin with the first claim. Because w is boundary expressible that is bounded by E, by Lemma 3.3.8, the $t-$error for any point (x, y) is given by (A.4). Maximizing $t^E(x, y)$ with respect to $m_w(x, y)$ results in a single maximizer $m_w(x, y) = 1$, and a maximum value: $t^E(x, y) \leq t^E(x, y)|_{m_w(x,y)=1} = 2(2 + 2B_w)^{-1/2}$. Note that the right hand side of this upper bound is itself bounded by t if and only if $B_w \geq 2t^{-2} - 1$. Therefore, if $(B_w \geq 2t^{-2} - 1)$, then $t^E(x, y) \leq t$ for each vertex $(x, y) \in w$. Further, if $k = \infty$, then by the density of the rationals in the reals, there will always exist vertices $(x, y) \in w$ with $m_w(x, y)$ arbitrarily close to 1. Thus if $B_w < (2t^{-2} - 1)$ and $k = \infty$, there will exist vertices in w that are not $t-$reachable using E.

To prove the second claim of the Lemma, we solve $t^E(x, y) \leq t$ (with t^E given in (A.4)) for $m_w(x, y)$ which yields the result of the Lemma. □

A.5 Lemma 3.3.12

Lemma A.5.1 (BEW Closed Under Splitting)**.** *Let $w = W[(x_1, y_1), (x_2, y_2)]$ be a BEW that is bounded by a control set E for a lattice L^n. The wedges $w_1 = W[(x_1, y_1), v_w^1]$ and $w_2 = W[v_w^1, (x_2, y_2)]$ are both BEWs that are bounded by $E \cup \{v_w^1\}$, and $w_1 \cup w_2 = w$. Further, if w is upper or lower separable, then the wedges $w_{unreach}, w_{between}$ are BEWs and $w_{unreach}$ is bounded by $E \cup \{v_w^k\}$.*

Proof. We begin by showing that w_1, w_2 are both BEWs. Observe that $w_1 \cup w_2 = w$ holds from the definition of a wedge. Because w is a BEW, it must hold that for any $u \in w$ there exists natural numbers k_1, k_2 with $u = (k_1x_1 + k_2x_2, k_1y_1 + k_2y_2)$. Assume without loss of generality, that $k_1 \leq k_2$ (a similar argument to the one that follows can be made if $k_1 > k_2$), and note that in this case, $u \in w_2$. Indeed, in this case, $m_w(u) = k_2l_2(k_1l_1)^{-1} \geq l_2(l_1)^{-1} = m_w(x_1 + x_2, y_1 + y_2)$ implying by Lemma 3.3.7, that u lies above the line passing through $(x_1 + x_2, y_1 + y_2)$ that is, in w_2. Then

$$u = k_1(x_1, y_1) + k_2(x_2, y_2) = k_1v_w^1 + (k_2 - k_1)(x_2, y_2).$$

By assumption, $k_2 - k_1 = k_3 \in \mathbb{N}$, thus $u = k_1v_w^1 + k_3(x_2, y_2)$ implying that there exists natural numbers k_1, k_3 such that u can be expressed as a linear combination of the boundary vertices of w_2. If $k_1 > k_2$, then $u \in w_1$, and a similar argument can be made to show that u can be expressed as a linear combination of the boundary vertices of w_1. Therefore both w_1, w_2 are BEWs. Since E contains no vertices in the interior of w, and $w = w_1 \cup w_2$, it must hold that $E \cup \{v_w^1\}$ contains no vertices in the interior of w_1 or w_2. Thus w_1, w_2 are both BEWs bounded by $E \cup \{v_w^1\}$.

Next we show that if w is upper or lower separable, then the wedges $w_{\text{unreach}}, w_{\text{between}}$ are BEWs. We reserve our argument to the case of an upper separable wedge for brevity as a near identical argument can be made if w is lower separable using E. We prove these claims for an upper separable wedge by way of induction on k. As a base case, if $k = 1$, then $w_{\text{unreach}}, w_{\text{between}}$ are equal to w_1, w_2 and the result holds from the above argument. Assume now that the result holds for some $k - 1 \geq 2$. Letting $w' = W[(x_1, y_1), v_w^{k-1}]$, we observe that w' is a BEW by the induction assumption. Therefore, by the base case, the wedges $w_1' = W[(x_1, y_1), v_w^{k-1} + (x_1, y_1)], w_2' = W[v_w^{k-1} + (x_1, y_1), v_w^{k-1}]$ are both BEWs. Noting that $w_1' = W[(x_1, y_1), v_w^k] = w_{\text{unreach}}, w_2' = W[v_w^k, v_w^{k-1}] = w_{\text{between}}$ by (3.9) completes the proof that $w_{\text{unreach}}, w_{\text{between}}$ are BEWs. Finally, because w is bounded by E, E has no vertices in the interior of w by definition. Therefore, E has no vertices in the interior of w_{unreach} implying that w_{unreach} is bounded by $E \cup \{v_w^k\}$. □

A.6 Lemma 3.3.13

Lemma A.6.1. *Let $t \in (1, \sqrt{2})$, and let $w = W[(x_1, y_1), (x_2, y_2)]$ be a BEW that is bounded by a control set E for a lattice L^n. If w is upper (or lower) separable using E, then*

$$B_{w_{between}} \geq \frac{1}{t}. \tag{A.9}$$

Proof. The proof will be presented for the case when w is upper separable as an almost identical argument can be made if w is lower separable using E. Begin by observing that w_{between} is a BEW by Lemma 3.3.12. Note that $B_w > 0$ for any wedge other than $W[(0, 1), (1, 0)]$. Furthermore, if $t < \sqrt{2}$, then this lemma does not apply to this wedge. Therefore, we can conclude that if (A.9) does not hold, then

$$t < \frac{1}{B_{w_{\text{between}}}}.$$

For simplicity of notation, let $v_1 = (x_1, y_1), v_2 = (x_2, y_2), v_3 = v_w^{k-1}, v_4 = v_w^k$, and let $l_i = ||v_i||, i = 1, 2, 3, 4$. Then, by definition of the upper separability, it must hold that $m_w(v_3) \geq \overline{m}_w$, implying by Lemma 3.3.9 that v_3 is $t-$reachable using E. Therefore, by equation (A.4) of Lemma 3.3.8:

$$t^E(v_3) = \frac{1 + m_w(v_3)}{\sqrt{1 + m_w(v_3)^2 + 2m_w(v_3)B_w}} \leq t < \frac{1}{B_{w_{\text{between}}}}. \tag{A.10}$$

By the definition of the inner product,

$$B_w = \frac{v_1 \cdot v_2}{l_1 l_2} = \frac{x_1 x_2 + y_1 y_2}{l_1 l_2},\ B_{w_{\text{between}}} = \frac{v_3 \cdot v_4}{l_3 l_4}. \tag{A.11}$$

Replacing $m_w(v_3) = l_2((k-1)l_1)^{-1}$, v_i, l_i, and B_w in $B_{w_{\text{between}}}$ from (A.11) yields

$$B_{w_{\text{between}}} = \frac{(k-1)m_w(v_3)^2 + m_w(v_3)B_w(2k-1) + k}{\sqrt{(2m_w(v_3)B_w + m_w(v_3)^2 + 1)\,(2m_w(v_3)B_w k(k-1) + k^2 + m_w(v_3)^2(k-1)^2)}}.$$

Observe that $B_{w_{\text{between}}}$ given here is an increasing function of k for all $B_w \leq 1$. Further, observe that $k \geq 2$. Indeed, if $k = 1$, then $v_4 = v_w^1$. However, v_4 is not $t-$reachable by the definition of the upper separability factor. Therefore, if $v_4 = v_w^1$ is not $t-$reachable, then by Lemma 3.3.9, w is shortest separable, not upper separable which is a contradiction. Therefore, since $B_{w_{\text{between}}}$ is an increasing function of k, and $k \geq 2$, it must hold that $B_{w_{\text{between}}} \geq B_{w_{\text{between}}}|_{k=2}$. Thus by (A.10),

$$\begin{aligned}
&\frac{1 + m_w(v_3)}{\sqrt{1 + m_w(v_3)^2 + 2m_w(v_3)B_w}} < \frac{1}{B_{w_{\text{between}}}} \leq \frac{1}{B_{w_{\text{between}}}|_{k=2}} \\
=&\frac{\sqrt{(2B_w m_w(v_3) + m_2(v_3)^2 + 1)\,(4B_2 m_w(v_3) + m_w(v_3)^2 + 4)}}{3B_w m_w(v_3) + m_w(v_3)^2 + 2} \\
\implies& 2m_w(v_3)^4 + (11B_w - 1)m_w(v_3)^3 + (18B_w^2 - 2B_w + 8)m_w(v_3)^2 \\
&+ (23B_w - 1)m_w(v_3) + 8 \leq 0.
\end{aligned}$$

Observe now that $m_w(v_3) \geq \overline{m}_w$ by the definition of the separability factor and $\overline{m}_w \geq 1$ by Remark 3.3.10. Therefore, $m_w(v_3) \geq 1$. Observe that the left hand side of the final inequality above is an increasing function of B_w for all $m_w(v_3) \geq 1$. Therefore, since $B_w \geq 0$, we can conclude that

$$2m_w(v_3)^4 - m_w(v_3)^3 + 8m_w(v_3)^2 - m_w(v_3) + 8 \leq 0,$$

which cannot happen if $m_w(v_3) \geq 1$. Therefore, the assumption $t < B_{w_{\text{between}}}^{-1}$ results in a contradiction. □

A.7 Lemma 3.3.14

Lemma A.7.1. *Let $w = W[(x_1, y_1), (x_2, y_2)]$ be a BEW that is bounded by a control set E on a lattice L^n. If w is not deletable using E, let v denote the shortest non $t-$reachable vertex in w using E. The the following hold:*

1. *If w is shortest separable using E, then $v_w^1 = v$.*

2. *If w is either upper or lower separable using E and v_w^k is not $t-$reachable using E, then $v = v_w^k$ and every vertex in $w_{reach} \cup w_{between}$ is $t-$reachable using $E \cup \{v_w^k\}$.*

3. *If w is either upper or lower separable using E and v_w^k is $t-$reachable using E, then w is deletable using $E \cup \{v\}$.*

Proof. We prove the claims in the order that they appear. If w is shortest separable using E, then by definition of shortest separability, $m_w(v_w^1) \in (\underline{m}_w, \overline{m}_w)$ which by Lemma 3.3.9 implies that v_w^1 is not $t-$reachable using E. Observe that v_w^1 is the shortest vertex in the interior of w. Indeed, if u is a vertex lying in the interior of w, then by definition of the interior of a BEW, $u = k_1(x_1, y_1) + k_2(x_2, y_2)$ for $k_1, k_1 \in \mathbb{N}_{\geq 1}$. Thus $||u||$ is minimized for $k_1 = k_2 = 1$ resulting in $u = v_w^1$. Finally observe that any vertex lying on the boundary of w – that is, any vertex u of the form above where $k_1 = 0 \vee k_2 = 0$ – is trivially $t-$reachable using E for any $t \geq 1$. Indeed, it is assumed that w is bounded by E, thus $(x_1, y_1), (x_2, y_2) \in E$ implying that $d^E(u) = ||u||$. Therefore, v_w^1 is the shortest non $t-$reachable vertex in w using E establishing the first claim.

We prove the second claim for the case that w is upper separable as a near identical argument can be made if w is lower separable using E. We begin by showing that v_w^k is the shortest such vertex in w. Suppose that $u \in w$ is the shortest non $t-$reachable vertex in w. Since w is a BEW, there must exist natural numbers k_1, k_2 with $u = (k_1x_1 + k_2x_2, k_1y_1 + k_2y_2)$. Observe that if $u \in w$ does not lie in the interior of w, then u is trivially $t-$reachable using E. Therefore, if u is not $t-$reachable using E, then u must lie in the interior of w implying that $k_1, k_2 \geq 1$. Suppose that $k_1 \leq k - 1$ and observe that $m_w(v_w^{k-1}) = l_2((k-1)l_1)^{-1} \geq \overline{m}_w$ by the definition of the upper separability factor. Then $m_w(u) = k_2l_2(k_1l_1)^{-1} \geq l_2((k-1)l_1)^{-1} \geq \overline{m}_w$ implying by Lemma 3.3.9, that u is $t-$reachable using E. Therefore, if u is not $t-$reachable using E, then $k_1 \geq k$. Minimizing $||u||$ given that $u = (k_1x_1 + k_2x_2, k_1y_1 + k_2y_2)$ over $k_1 \geq k, k_2 \geq 1$ results in $k_1 = k, k_2 = 1$ implying that $v_w^k = u$. Therefore, v_w^k is the shortest non $t-$reachable vertex in w using E. Next we show that every vertex in $w_{\text{reach}} \cup w_{\text{between}}$ is $t-$reachable using $E \cup \{v_w^k\}$. Suppose $u \in w_{\text{reach}}$. Then by definition of w_{reach} in (3.9), u must lie below the line passing through the origin and v_{k-1}. Therefore, by Lemma 3.3.7, $m_w(u) \geq m_w(v_w^{k-1})$. However, $m_w(v_w^{k-1}) \geq \overline{m}_w$ by the definition of the upper separability factor implying that $m_w(u) \geq \overline{m}_w$. Therefore, u is $t-$reachable using E (and therefore using $E \cup \{v_w^k\}$) by Lemma 3.3.9.

Assume now that $u \in w_{\text{between}}$. Note that the wedge w_{between} is a BEW by Lemma 3.3.12, and that the lower boundary vertex v_w^k of w_{between} is in $E \cup \{v_w^k\}$. Further, since $m_w(v_w^{k-1}) \geq \overline{m}_w$ by the definition of the upper seperability factor, v_w^{k-1} is $t-$reachable

using E (and therefore using $E \cup \{v_w^k\}$) by Lemma 3.3.9. Therefore, the upper boundary vertex, v_w^{k-1} of w_{between} is $t-$reachable using $E \cup \{v_w^k\}$. By equation (A.5) of Lemma 3.3.8, the $t-$error of $u \in w_{\text{between}}$ using $E \cup \{v_w^k\}$ is bounded by

$$t^{E\cup\{v_w^k\}}(u) \leq \frac{1 + t m_{w_{\text{between}}}(u)}{\sqrt{1 + m_{w_{\text{between}}}(u)^2 + 2m_{w_{\text{between}}}(u) B_{w_{\text{between}}}}}.$$

The right-hand-side of the above inequality is a decreasing function of $B_{w_{\text{between}}}$. Further, by Lemma 3.3.13, $B_{w_{\text{between}}} \geq t^{-1}$. Therefore, replacing this bound on $B_{w_{\text{between}}}$, we obtain

$$t^{E\cup\{v_w^k\}}(u) \leq \frac{\sqrt{t}(t m_{w_{\text{between}}}(u) + 1)}{\sqrt{t m_{w_{\text{between}}}(u)^2 + 2m_{w_{\text{between}}}(u) + t}},$$

the right-hand-side of which is an increasing function of $m_{w_{\text{between}}}(u)$. Therefore,

$$t^{E\cup\{v_w^k\}}(u) \leq \lim_{m_{w_{\text{between}}}(u)\to\infty} \frac{\sqrt{t}(t m_{w_{\text{between}}}(u) + 1)}{\sqrt{t m_{w_{\text{between}}}(u)^2 + 2m_{w_{\text{between}}}(u) + t}} = t,$$

implying that u is $t-$reachable using $E \cup \{v_w^k\}$.

We now prove the third claim. Again, we offer a proof for the case that w is upper separable as an almost identical argument can be made if w is lower separable using E. Observe that $m_w(v_w^k) \leq \underline{m}_w$. Indeed, $m_w(v_w^k) \leq \overline{m}_w$ by the definition of the upper separability factor. Therefore, if $m_w(v_w^k) > \underline{m}_w$, then $m_w(v_w^k) \in (\underline{m}_w, \overline{m}_w)$ implying that v_w^k is not $t-$reachable using E by Lemma 3.3.9. This contradicts the assumption that v_w^k is $t-$reachable using E. Therefore, $m_w(v_w^k) \leq \underline{m}_w$. This implies that the set of all vertices in w that are not $t-$reachable using E is a subset of w_{between}. Indeed, if u is not $t-$reachable using E then $m_w(u) < \overline{m}_w$. Since $m_w(v_w^{k-1}) \geq \overline{m}_w$ by definition of the upper separability factor, it must hold that u lies below the line passing through the origin and v_w^{k-1} by Lemma 3.3.7. Further, since $m_w(u) > \underline{m}_w$, u must lie above the line passing through the origin and v_w^k. Thus $u \in w_{\text{between}}$ and the set of all vertices that are not $t-$reachable using E is a subset of w_{between}. Therefore, $v \in w_{\text{between}}$ and there exists natural numbers $k_1, k_2 > 1$ such that $v = k_1 v_w^k + k_2 v_w^{k-1}$ because w_{between} is a BEW by Lemma 3.3.12. Let $v_1 = (k_1 - 1)v_w^k + v_w^{k-1}, v_2 = v_w^k + (k_2 - 1)v_w^{k-1}$. We make several observations:

1. The wedges $w_1 = W[v_1, v], w_2 = W[v, v_2]$ are BEWs.
2. The vertices v_1, v_2 are $t-$reachable using E.
3. Every vertex in $w - w_1 - w_2$ is $t-$reachable using E.

4. It holds that $B_{w_1}, B_{w_2} \geq B_{w_{\text{between}}}$.

To show the first observation, note that the wedge $w_1 \cup w_2 = W[v_1, v_2]$ is a BEW. Indeed, by repeated application of the result of Lemma 3.3.12, the wedge $W[v_w^k, v_1]$ is a BEW because w_{between} is. Moreover, by repeated application of the result of Lemma 3.3.12, $W[v_1, v_2]$ is a BEW because $W[v_w^k, v_1]$ is. Finally, noting that $v = v_1 + v_2$, we may conclude that w_1, v_2 are both BEWs by Lemma 3.3.12. Next, observe that $||v_1||, ||v_2|| < ||v|$ by construction. Therefore, since v is the shortest non $t-$reachable vertex in w using E, it must hold that v_1, v_2 are $t-$reachable using E and the second observation holds. Observe that v_1 lies below the line passing through the origin and v, implying by Lemma 3.3.7, that $m_w(v_1) < m_w(v)$. Similarly, $m_w(v_2) > m_w(v)$. By the second observation, if $m_w(v_1) < m_w(v)$ then $m_w(v_1) \leq \underline{m}_w$. Indeed, if this were not the case then $\underline{m}_w < m_w(v_1) < m_w(v) < \overline{m}_w$ implying that v_1 would not be $t-$reachable using E by Lemma 3.3.9. Similarly, $m_w(v_2) \geq \overline{m}_w$. If $u \in w - w_1, w_2$, then either $m_w(u) \geq m_w(v_2) \geq \overline{m}_w$ or $m_w(u) \leq m_w(v_1) \leq \underline{m}_w$ implying that u is $t-$reachable using E in either case by Lemma 3.3.9. Therefore, the third observation holds. Finally, observe that $w_1, w_2 \subset w_{\text{between}}$ by construction, implying that $B_{w_1}, B_{w_2} \geq B_{w_{\text{between}}}$ and the last observation holds.

By the third observation, to show that every vertex $u \in w$ is $t-$reachable using $E \cup \{v\}$, it suffices to show that every vertex $u \in w_1 \cup w_2$ is $t-$reachable using $E \cup \{v\}$. Let $u \in w_1 \cup w_2$. Because these wedges are BEWs by the first observation, and v_1, v_2 are $t-$reachable using E (and therefore using $E \cup \{v\}$) by the second observation, it must hold by (A.5), (A.6) of Lemma 3.3.8, that

$$\begin{aligned} t^{E\cup\{v\}}(u) &\leq \frac{t + m_{w_1}(u)}{\sqrt{1 + m_{w_1}(u)^2 + 2m_{w_1}(u)B_{w_1}}}, \ \forall u \in w_1, \\ t^{E\cup\{v\}}(u) &\leq \frac{1 + tm_{w_2}(u)}{\sqrt{1 + m_{w_2}(u)^2 + 2m_{w_2}(u)B_{w_2}}}, \ \forall u \in w_2. \end{aligned} \tag{A.12}$$

By the fourth observation and Lemma 3.3.13, $B_{w_1}, B_{w_2} \geq t^{-1}$. Replacing these bounds in (A.12) results in new bounds that are increasing functions of $m_{w_1}(u), m_{w_2}(u)$, respectively. Taking the limit as $m_{w_1}, m_{w_2} \to \infty$ yields t in both cases. Therefore, we can conclude that $t^{E\cup\{v\}}(u) \leq t$ and u is $t-$reachable for any $u \in w_1 \cup w_2$ using $E \cup \{v\}$. □

A.8 Lemma 3.3.15

Lemma A.8.1. *If $v \in L^n$ is not $t-$reachable using E^r, then v lies in a BEW that is bounded by E^r for any $r \geq 0$.*

Proof of Lemma 3.3.15. We offer a proof by way of induction on r. As a base case, if $r = 0$, then E^0 is given by Line 2 of the algorithm. Observe that we may form a set of BEWs $\mathcal{W} = \{W[(1,0),(0,1)], W[(0,1),(-1,0))], W[(-1,0),(0,-1)], W[(0,-1),(1,0)]\}$ each of which is bounded by E^0 and $\bigcup_{w\in\mathcal{W}} w = L^n$. Therefore, if $v \in L^n$ is not $t-$reachable using E^0, then $v \in \bigcup_{w\in\mathcal{W}} w$ is in a BEW that is bounded by E^0. Assume that the result holds at iteration r for any $r \geq 0$. Observe that $E^{r+1} = E^r \cup \{v\}$ where v is the shortest (ties broken arbitrarily) vertex in L^n that was not $t-$reachable using E^r. By the induction assumption $v \in w = W[(x_1, y_1), (x_2, y_2)]$ for a BEW that is bounded by E^r. Suppose now that $u \in L^n$ is not $t-$reachable using E^{r+1}. If $u \notin w$, then the result holds trivially by the induction assumption. Otherwise, if $u \in w$, let $w_1 = W[(x_1, y_1), v], w_2 = W[v, (x_2, y_2)]$ then $u \in w_1 \cup w_2$. We consider three exhaustive cases: w is deletable, shortest separable, or upper/lower separable using E^r. If w is deletable using E^r, then no such $u \in w$ can exist which is a contradiction. If w is shortest separable using E^r, then $v = v_w^1$ by Lemma 3.3.14. In this case, both w_1, w_2 are BEWs bounded by E^{r+1} by Lemma 3.3.12, and $u \in w_1 \cup w_2$ lies in a BEW that is bounded by E^{r+1}. Finally, if w is either upper or lower separable using E^r, then there are two sub-cases: first, if $m_w(v_w^k) \notin (\underline{m}_w, \overline{m}_w)$, then by the third result of Lemma 3.3.14, no u in w can exist that is not $t-$reachable using E^{r+1}, which is a contradiction. If $m_w(v_w^k) \in (\underline{m}_w, \overline{m}_w)$, then $v = v_w^k$ by the second result of Lemma 3.3.14. Further, by the same result $u \in w_{\text{unreach}}$. By Lemma 3.3.12, w_{unreach} is a BEW that is bounded by $E^r \cup \{v_w^k\} = E^{r+1}$. Therefore, in all three cases, if $u \in w$ is not $t-$reachable using E^{r+1} then u is in a BEW that is bounded by E^{r+1}. □

A.9 Lemma 3.3.16

Lemma A.9.1. *Let $w = W[(x_1, y_1), (x_2, y_2)]$ be a BEW that is bounded by a control set E. If w is either upper or lower separable using E, let θ denote the angle inscribed by the wedge $w_{between} \cup w_{reach}$. Then*

$$\cos(\theta) \geq \frac{2}{t^2} - 1. \tag{A.13}$$

Further, if $m_w(v_w^k) \notin (\underline{m}_w, \overline{m}_w)$, let v denote the shortest vertex in w that is not $t-$reachable using E. Then

$$\min(\cos(\theta_1), \cos(\theta_2)) \geq \frac{2}{t^2} - 1, \tag{A.14}$$

where θ_1, θ_2 are the angles inscribed by the wedges $w_1 = W[(x_1, y_1), v], W[v, (x_2, y_2)]$, respectively.

Proof. We prove this results for the case where w is upper separable using E, an identical argument can be made for the case when w is lower separable. By the cosine law, the angle inscribed by the wedge w_{reach} is

$$\cos(\theta_{\text{reach}}) = \frac{||v_w^{k-1}||^2 + l_2^2 - (k-1)^2 l_1^2}{2||v_w^{k-1}||l_2}$$

where $l_1 = ||x_1, y_1||, l_2 = ||x_2, y_2||$. Observing that $\cos(\theta_{\text{reach}})$ is an increasing function of $||v_w^{k-1}||$ and $t||v_w^{k-1}|| \geq d^E(v_w^{k-1}) = (k-1)l_1 + l_2$ because v_w^{k-1} is $t-$reachable using E by the definition of the upper separability factor, k, it must hold that

$$\cos(\theta_{\text{reach}}) \geq \frac{t^2(m_w(v_w^{k-1}) - 1) + m_w(v_w^{k-1}) + 1}{2(m_w(v_w^{k-1}))^2 t},$$

where, by definition, $m_w(v_w^{k-1}) = l_2((k-1)l_1)^{-1}$. Observing that the right hand side of the above is an increasing function of $m_w(v_w^{k-1})$ and $m_w(v_w^{k-1}) \geq \overline{m}_w$ by the definition of the upper separability factor, we can conclude that

$$\cos(\theta_{\text{reach}}) \geq \frac{t^2(\overline{m}_w - 1) + \overline{m}_w + 1}{2\overline{m}_w^2 t}.$$

Further, by Lemma 3.3.13, $\cos(\theta_{\text{between}}) \geq t^{-1}$ where θ_{between} is the angle inscribed by the wedge θ_{between}. Therefore,

$$\begin{aligned}\cos(\theta) &= \cos(\theta_{\text{reach}} + \theta_{\text{between}}) \\ &\geq \cos\left(\arccos\left(\frac{t^2(\overline{m}_w - 1) + \overline{m}_w + 1}{2\overline{m}_w^2 t}\right) + \arccos\left(\frac{1}{t}\right)\right).\end{aligned}$$

Replacing the definition of $\overline{m}_w$ in (A.8) in the above results in a non-increasing function of B_w. Since w is upper separable and not deletable, it must hold by the first result in Lemma 3.3.9, that $B_w < 2t^{-2} - 1$. Replacing this last in the result yields exactly $2t^{-2} - 1$ implying that (A.13) holds.

If $m_w(v_w^k) \notin (\underline{m}_w, \overline{m}_w)$, then v_w^k is $t-$reachable using E, and it must hold that $m_w(v_w^k) \leq \underline{m}_w$ since $m_w(v_w^k) \leq \underline{m}_w$ by the definition of the upper separability factor, and $m_w(v_w^k) \notin (\underline{m}_w, \overline{m}_w)$ by assumption. This implies that $m_w(v) \in (m_w(v_w^k), m_w(v_w^{k-1})))$ since v is not $t-$reachable using E (implying that $m_w(v) \in (\underline{m}_w, \overline{m}_w)$ by Lemma 3.3.9), $m_w(v_w^{k-1}) \geq \underline{m}_w$ (by the definition of the upper separability factor), and $m_w(v_w^k) \leq \underline{m}_w$. Therefore, v must lie between the lines connecting the origin and v_w^k and connecting the origin and v_w^{k-1}. Therefore, $\theta_1 \leq \theta_{\text{unreach}} + \theta_{\text{between}}, \theta_2 \leq \theta_{\text{reach}} + \theta_{\text{between}}$ where θ_{unreach} is the angle inscribed by the wedge w_{unreach}. It has already been established that $\cos(\theta_{\text{reach}} + \theta_{\text{between}}) \geq 2t^{-2} - 1$. Further, using an almost identical argument, it is easily verified that $\cos(\theta_{\text{unreach}} + \theta_{\text{between}}) \geq 2t^{-2} - 1$ using the fact that $m_w(v_w^k) \leq \underline{m}_w$ and $kl_1 + l_2 \leq t||v_w^k||$ by Lemma 3.3.9. □

A.10 Lemma 3.3.17

Lemma A.10.1 (Distance Invariance)**.** *Let $t \in (1, \sqrt{2})$, and let E^r denote the control set at the beginning of iteration r of the for loop in Line 3 of Algorithm 3. Let $w = W[(x_1, y_1), (x_2, y_2)]$ be a BEW wedge that is bounded by E^r. Then the following are true:*

1. *If w is shortest separable using E^r, then for any other control set E' with $v_w^1 \notin E'$, it holds that $d^{E'}(v_w^1) \geq d^{E^r}(v_w^1)$.*
2. *If w is upper or lower separable using E^r and $m_w(v_w^k) \in (\underline{m}_w, \overline{m}_w)$, then for all vertices $u \in w_{unreach}, u \neq v$, $||u|| + ||v_w^k - u|| \geq d^{E^r}(v_w^k)$.*

Proof. We prove the claims in the order in which they appear. Let E' be a control set with $v_w^1 \notin E'$. We offer a proof by way of contradiction. Suppose that $d^{E'}(v_w^1) < d^{E^r}(v_w^1)$. Then there must exist a vertex $u \in L^n$ such that $u \in E', u \notin E^r$ and such that the path using E' to v_w^1 contains the relative motion u. Further, by the triangle inequality, $d^{E'}(v_w^1) \geq ||u|| + ||v_w^1 - u||$. Therefore, if $E'' = \{u, v_w^1 - u, (x_1, y_1), (x_2, y_2)\}$, then $d^{E'}(v_w^1) \geq d^{E''}(v_w^1)$. Without loss of generality, we may assume that $u \in w$. Indeed, by Lemma 3.3.6, $u \in w \vee v_w^1 - u \in w$ otherwise a path in E'' to v_w^1 will not contain either of the relative motions $u, v_w^1 - u$ which contradicts our assumption. Therefore we declare u to be the vertex in w. Note further that if u is on the boundary of of w, then it trivially holds that $d^{E''}(v_w^1) \geq d^{E^r}(v_w^1)$ and a contradiction is reached. Therefore, we may assume that u lies in the interior of w, implying that there must exist $k_1, k_2 \in \mathbb{N}_{\geq 1}$ such that $u = (k_1 x_1 + k_2 x_2, k_1 y_1 + k_2 y_2)$ by the definition of the interior of a BEW. Therefore,

$$\begin{aligned} d^{E''}(v_w^1) =& ||u|| + ||v_w^1 - u|| \\ =& ||k_1 x_1 + k_2 x_2, k_1 y_1 + k_2 y_2|| + \\ & ||(1 - k_1)x_1 + (1 - k_2)x_2, (1 - k_1)y_1 + (1 - k_2)y_2||. \end{aligned}$$

Assume (without loss of generality), that $||x_1, y_1|| = l_1 \leq ||x_2, y_2|| = l_2$ (an identical proof exists if this is not the case). Then, the right hand side of the above inequality is minimized for $k_1, k_2 \geq 1, u \neq v_w^1$, by $k_1 = 2, k_2 = 1$ yielding

$$d^{E''}(v_w^1) \geq ||2x_1 + x_2, 2y_1 + y_2|| + l_1.$$

By Lemma 3.3.6, $d^{E^r}(v_w^1) = l_1 + l_2$. Therefore, recalling that $d^{E'}(v_w^1) \geq d^{E''}(v_w^1)$,

$$\begin{aligned} d^{E'}(v_w^1) \geq d^{E^r}(v_w^1) \impliedby & ||2x_1 + x_2, 2y_1 + y_2|| + l_1 \geq l_1 + l_2 \\ \iff \sqrt{(2x_1 + x_2)^2 + (2y_1 + y_2)^2} \geq l_2 \impliedby & \sqrt{4l_1^2 + l_2^2} \geq l_2 \end{aligned}$$

the last line of which holds for all $l_1, l_2 \geq 0$. Therefore, $d^{E'}(v_w^1) \geq d^{E^r}(v_w^1)$ as desired.

We now prove the second claim. We prove this claim for upper separable wedges using E but omit proof for lower separable wedges as these proofs are almost identical. Therefore, we must show that if w is an upper separable wedge using E^r and $m_w(v_w^k) \in (\underline{m}_w, \overline{m}_w)$ then for all $u \in w_{\text{unreach}}$, $d^{E^r}(v_w^k) \leq ||u|| + ||v_w^k - u||$. By Lemma 3.3.15, each vertex that is not $t-$reachable using E^r on iteration r of the for loop in Line 3 of 3 lies in a boundary expressible wedge that is bounded by E^r. Letting $\mathcal{W}^r$ denote the set of all such wedges w' that are upper separable using E^r and such that $v_{w'}^k \in (\underline{m}_{w'}, \overline{m}_{w'})$, we observe that $w \in \mathcal{W}$. To prove the result, we make the following interim claim: For each $w' = W[(x_1', y_1'), (x_2', y_2')] \in \mathcal{W}^r$,

$$|y_2'||x_1'| \leq |x_2' y_1'| + y_1^{'2} + 1. \tag{A.15}$$

For simplicity, we limit $\mathcal{W}^r$ to wedges in the first quadrant of $\mathbb{R}^2$. The assumption does not alter the proof of the interim claim. We prove the validity of this interim claim by induction on r. If $t < \sqrt{2}$, then when $r = 0$, $\mathcal{W}^0 = \{W[(1,0),(0,1)]\}$, and we observe that (A.15) reduces to $1 \leq 0 + 0^1 + 1$. As an induction hypothesis, assume that (A.15) holds for some $r \geq 0$ for an arbitrary wedge $w' \in \mathcal{W}^r$. At iteration $r+1$ of the for loop in Line 3, let v denote the vertex that is added to E^r in line 5. If $v \notin w'$ then $w' \in \mathcal{W}^{r+1}$ remains unchanged and (A.15) continues to hold. If, however, $v \in w'$ then $w' \notin \mathcal{W}^{r+1}$ since w' is not bounded by E^{r+1}. By the definition of $\mathcal{W}^r$, $v_{w'}^k \in (\underline{m}_{w'}, \overline{m}_{w'})$ implying that $v_{w'}^k$ is not $t-$reachable using E^r (Lemma 3.3.9) and $v_{w'}^k$ is the shortest non $t-$reachable vertex in w' using E^r (Lemma 3.3.14). Therefore, since v is the shortest vertex in L^n (and therefore in w') that is not $t-$reachable using E^r, it must hold that $v = v_{w'}^k$. Thus at the beginning of iteration $r+1$, w' is split into two sub-wedges $w_1' = W[(x_1', y_1'), v_{w'}^k], w_2' = W[v_{w'}^k, (x_2, y_2)]$. Observe that $w_2' \notin \mathcal{W}^{r+1}$ since the length of its lower boundary vertex exceeds that of its upper implying that w_2' cannot be upper separable using E^{r+1} by the definition of upper separable wedges in Definition 3.3.11. If w_1' is also not upper separable using E^{r+1}, then interim claim holds vacuously on iteration $r+1$. Otherwise, (A.15) for the wedge w_1' reduces to $(ky_1' + y2')x_1' \leq x_1^{'2} + (kx_1' + x_2')y_1' + y_1^{'2}$ which holds if and only if $y_w' x_1' \leq x_1^{'2} + x_2' y_1' + y_1^{'2}$ which is the induction assumption. Therefore, for any wedge $w' = W[(x_1', y_1'), (x_2', y_2')] \in \mathcal{W}^r$ at any iteration r of the for loop in line 3 of Algorithm 3, the inequality (A.15) must hold.

As noted above, $w \in \mathcal{W}^r$ by construction of $\mathcal{W}^r$. We offer a proof of result of the Lemma by way of contradiction. For simplicity, we limit the discussion to the first quadrant of the plane since an equivalent argument can be made to show the result in all other quadrants. For the purposes of contradiction, assume that $u \in w_{\text{unreach}}$, $u \neq v_w^k$ but

$$||u|| + ||v_w^k - u|| < d^{E^r}(v_w^k). \tag{A.16}$$

Since w is a BEW bounded by E^r, it must hold that $d^{E^r}(v_w^k) = kl_1 + l_2$ where $||x_1, y_1|| = l_1, ||x_2, y_2|| = l_2$. Further, there must exist natural numbers k_1, k_2 such that $u = k_1(x_1, y_1) + k_2(x_2, y_2)$. Minimizing $||u|| + ||v_w^k - u||$ over all $u \in w_{\text{between}}, u \neq v_w^k$ yields $k_1 = k+1, k_2 = 1$ and $u = v_w^k + (x_1, y_1)$. Replacing this u in (A.16) yields

$$((B_w - 1)k + 1 + B_w)m_w(v_w^1) + 2k < 0, \tag{A.17}$$

by the law of cosines and the definition $m_w(v_w^1) = l_2(l_1)^{-1}$. Observe that (A.17) implies that $(B_w - 1)k + 1 + B_w < 0$ implying in turn that the left hand side of (A.17) is a decreasing function of $m_w(v_w^1)$ and an increasing function of B_w. Note B_w must be a decreasing function of y_2 since larger values of y_2 result in larger values of θ_w and therefore smaller values of $B_w = \cos(\theta_w)$. Further, $m_w(v_w^1)$ must be an increasing function of y_2 since l_2 increases with y_2 and $m_w(v^1) = l_2/l_1$. Therefore, the left hand side of (A.17) can be no smaller than when y_2 is at its maximum. By the interim claim, $y_2x_1 \leq x_2y_1 + y_1^2 + 1$. Replacing this constraint in (A.17) and simplifying observing that $B_w = (x_1x_2 + y_1y_2)(l_1l_2)^{-1}$ (by the definition of the inner product) yields $k \leq 1/3$ which is a contradiction. □

A.11 Lemma 3.3.18

Lemma A.11.1 (Base Case For Inductive Proof)**.** *Let $t > 1$ and L^n a lattice. Let $\mathcal{E}^t$ denote the set of all minimal $t-$spanning control sets of L^n, and let $E = \{(0,1),(1,0),(-1,0),(0,-1)\}$. If $t \geq \sqrt{2}$ then $E \in \mathcal{E}^t$. Furthermore, if $t < \sqrt{2}$ then $E \subseteq E^*$ for all $E^* \in \mathcal{E}^t$.*

Proof of Lemma 3.3.18. Note that $w = W[(1,0),(0,1)]$ is a BEW that is bounded by E. Therefore, by equation (A.4) of Lemma 3.3.8, the $t-$error of any point u in L^n is given by

$$t^E(u) = \frac{1 + m_w(u)}{\sqrt{1 + m_w(u)^2}}$$

because $B_w = \cos(\pi) = 0$. Maximizing $t^E(u)$ with respect to $m_w(u)$ yields $t^E(u) \leq \sqrt{2}$ implying that every vertex in w is $t-$reachable using E if $t \geq \sqrt{2}$. A similar argument can be made if u lies in the second, third, and fourth quadrant of $\mathbb{R}^2$. Further, there cannot be a smaller set E that $t-$spans L^n as this set would contain no more than 3 elements.

Further, there cannot be a smaller MTSCS as such a set would contain only one vertex which would fail to span (let alone $t-$span) $L^n \subset \mathbb{R}^2$. Suppose now that $t < \sqrt{2}$. Suppose

that $E^* \in \mathcal{E}^t$ with $E \not\subset E^*$. Suppose, without loss of generality, that $(0,1) \notin E^*$. Then the shortest path in E^* to $(0,1)$ is at least as long as the path

$$(1,1) + (-1,0).$$

Therefore,

$$R^{E^*}(0,1) \geq \frac{\sqrt{2}+1}{1} > \sqrt{2} > t.$$

which is a contradiction. □

Appendix B

Proofs of Results in Chapter 4

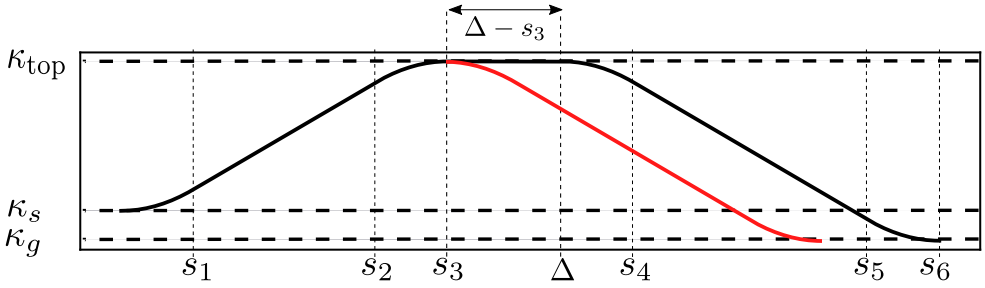


Figure B.1: Curvature profile of two curves in the set $\mathcal{G}$. (Red): curvature of the curve $G(\boldsymbol{p}_s, \kappa_{\text{top}}, \kappa_f, \Delta = s_3)$. (Black): curvature of the curve $G(\boldsymbol{p}_s, \kappa_{\text{top}}, \kappa_f, \Delta)$ for $\Delta > s_3$.

Proof of Lemma 4.4.1. Consider a set of curves $\mathcal{G}$. Let $G(\Delta) : \mathbb{R}_{\geq s_3} \to \mathcal{G}$ be the function that sends values of $\Delta \geq s_3$ to the corresponding G^3 curve in $\mathcal{G}$. Let $x(s, \Delta)$ $y(s, \Delta)$, $\theta(s, \Delta)$, $\kappa(s, \Delta)$ denote the path state profiles of the curve $G(\Delta)$ at arc-length s, and let $x_f(\Delta), y_f(\Delta), \theta_f(\Delta), \kappa_f(\Delta)$ denote the final path states of $G(\Delta)$. From Equations (4.1), (4.12), we make the following observation:

$$\theta(s, \Delta) = \begin{cases} \theta(s, s_3), & \text{if } s \leq s_3 \\ \theta(s_3, s_3) + \kappa_{\text{top}}(s - s_3), & \text{if } s_3 < s \leq \Delta \\ \kappa_{\text{top}}(\Delta - s_3) + \theta(s - (\Delta - s_3), s_3), & \text{if } \Delta < s \leq s_6. \end{cases} \tag{B.1}$$

As implied by Figure B.1, if s_6 is the final arc-length of the curve $G(\Delta)$, then $s_6 - (\Delta - s_3)$ is equal to the final arc-length of the curve $G(s_3)$. Therefore, we can conclude from (B.1)

that $\theta_f(\Delta) = \kappa_{\text{top}}(\Delta - s_3) + \theta_f(s_3)$. Setting this last to θ_g, and solving for Δ completes the proof. □

Proof Of Theorem 4.4.2. This proof uses the notation of the Proof of Lemma 4.4.1. Given a set of curves $\mathcal{G}$, it can be shown from equations (B.1), and (4.1), that

$$\begin{aligned}
\theta_f(\Delta) =& \theta_f(s_3) + \kappa_{\text{top}}(\Delta - s_3), \\
x_f(\Delta) =& x(s_3, s_3) - \frac{\sin(\theta(s_3)) - \sin(\kappa_{\text{top}}(\Delta - s_3) + \theta(s_3))}{\kappa_{\text{top}}} \\
&+ \cos(\kappa_{\text{top}}(\Delta - s_3))(x_f(s_3) - x(s_3, s_3)) \\
&- \sin(\kappa_{\text{top}}(\Delta - s_3))(y_f(s_3) - y(s_3, s_3)), \\
y_f(\Delta) =& y(s_3, s_3) - \frac{-\cos(\theta(s_3)) + \cos(\kappa_{\text{top}}(\Delta - s_3) + \theta(s_3))}{\kappa_{\text{top}}} \\
&+ \sin(\kappa_{\text{top}}(\Delta - s_3))(x_f(s_3) - x(s_3, s_3)) \\
&+ \cos(\kappa_{\text{top}}(\Delta - s_3))(y_f(s_3) - y(s_3, s_3)).
\end{aligned} \tag{B.2}$$

Observe now that the shortest distance from the point $\boldsymbol{x}_c = [x_c, y_c]^T$ given in (4.14) to the line passing through $(x_f(\Delta), y_f(\Delta))$ whose slope is $\tan\theta_f(\Delta)$, is given by

$$d(\Delta) = \frac{x_c \tan\theta_f(\Delta) - y_c + y_f(\Delta) - x_f(\Delta)\tan\theta_f(\Delta)}{\sqrt{\tan\theta_f(\Delta)^2 + 1}}.$$

Replacing (B.2) in $d(\Delta)$, yields a complete cancellation of the term Δ. The remaining parameters $\kappa_{\text{top}}, s_3, \boldsymbol{x}_c, etc.$, are identical for all members of $\mathcal{G}$, thus we can conclude that $d(\Delta)$ is the same for all members of $\mathcal{G}$ as desired. □

Glossary

G^3 A curve is G^3 continuous if it is thrice continuously differentiable with respect to arc-length. 61, 109

G^k A curve is G^k continuous if it is k-times continuously differentiable with respect to arc-length but not $k+1$-times continuously differentiable with respect to arc-length. 61

$t-$**error** A measure of the quality of a control set E. The worst case ratio of the distance using E to a lattice vertex to the cost of the direct motion to that vertex taken over all starts in a starting set and all lattice vertices. 36, 121

configuration A tuple of all of the states and inputs of a curve. 35, 59, 90, 118

configuration space The set of all possible configurations. 35, 91, 116

curve A k-times continuously differentiable function from an interval of the real line to the plane. 61, 109

Demonstrations A set of trajectories proposed by a user. Each trajectory is between known configurations. These trajectories can be obtained by monitoring a user as they drive for a fixed amount of time. We do not assume that a user is able to provide demonstrations that perfectly reflect their preferences. 9, 116

distance using E The cost of a path using E. 38, 95

global planner A global planner computes an initial trajectory between vehicles' start and goal configurations. This trajectory does not take the differential constraints of the vehicle into account. The initial trajectory can be expressed as a sequence of waypoints. The motion planning problem is then solved by the local planner between

global waypoints to produce a smooth trajectory that adheres to the differential constraints of the vehicle. 1

local planner A local planner obtains a goal configuration – possibly a waypoint in a larger motion – from the global planner as well as the current configuration of the vehicle, and solves the motion planning problem between configurations. The solution is used as a reference trajectory by the tracking controller. 1, 2

motion An equivalence class of trajectories. Here, the equivalence relation is defined as the set of all tuples of trajectories such that the first trajectory can be rotated and translated in $\mathbb{R}^2$ to produce the second trajectory. A motion is an abstraction of a trajectory that may be rotated or translated to connect different configurations. 35, 60, 90, 118

path A function from an interval of the real line representing arc-length to a configuration space. 36, 60

path using E A path using E from a starting vertex $o \in \mathcal{O}$ to a lattice vertex $j \in L$ is a minimal-cost path from o to j in the graph whose vertices are all of the lattice vertices, whose edges are pairs of lattice vertices for which there exists a motion in E taking the first to the second, and whose cost is the cost of those motions. 52

state An element that appears in the kinematic model of a curve but is not an input to the model. 64, 90

trajectory A function from an interval of the real line representing time to a configuration space. Uniquely defined by a path and a velocity profile. If the velocity profile has constant unit speed, then the trajectory is equal to the path. 59, 112, 118

velocity profile A function from an interval of the real line representing arc-length to the real line. Represents a rate of change of arc-length with respect to time. 60, 107

workspace A subset of $\mathbb{R}^2$. Typically used to define the boundaries of a motion planning problem and/or a bounding box for a lattice. 35, 92, 116