UNIVERSITY OF CALIFORNIA, MERCED

**Lane Determination and Optimization for Multi-Agent Systems**

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering and Computer Science
in the Graduate Division of
the University of California, Merced

by

Maryam Khazaei Pool

Committee in charge:

Professor Sungjin Im, Chair
Professor Marcelo Kallmann
Professor Shawn Newsam
Professor David Noelle

Fall 2023

The dissertation of Maryam Khazaei Pool is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

(Professor Marcelo Kallmann)

(Professor Shawn Newsam)

(Professor David Noelle)

(Professor Sungjin Im, Chair)

University of California, Merced

Fall 2023

DEDICATION

To my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my PhD supervisor, Professor Marcelo Kallmann, for his unwavering support, guidance, and expertise throughout this doctoral journey. His insightful feedback and encouragement have been invaluable in shaping the direction of my research.

I would like to convey my sincere thanks to my PhD advisor Professor Sungjin Im, for his encouragement, expertise throughout this doctoral journey and guidance. I appreciate the collaborative efforts of both mentors, and I am fortunate to have such dedicated and inspiring individuals supporting my academic endeavors.

I extend my sincere gratitude to committee members Professor Sungjin Im, Professor David Noell, and Professor Shawn Newsam for generously dedicating their time, offering invaluable guidance, and demonstrating remarkable patience during both my qualifying examination and thesis defense. Their expertise and constructive feedback have significantly enriched the quality of my research.

In this moment of accomplishment, I wish to extend my deepest appreciation to my husband Matthew. His belief in my abilities has been a constant source of motivation. I extend my appreciation to my family specially my mother and my father for their unconditional love and encouragement.

This dissertation would not have been possible without the financial assistance provided by Teaching Fellow and Teaching Assistant from University of California, Merced. I am sincerely thankful for the support from Professor Sungjin Im, Professor Shawn Newsam, and Professor Cerpa which allowed me to focus on my studies and research.

## VITA

| | |
|---|---|
| 2013 | B. S. in Applied Mathematics *cum laude*, Iran University of Science and Technology (IUST), Tehran |
| 2018 | M. S. in Applied Mathematics *cum laude*, University of Colorado, Denver |
| 2023 | Ph. D. in Electrical Engineering and Computer Science, University of California, Merced |

## PUBLICATIONS

Maryam Khazaei, Carlos Alvarenga, Marcelo Kallmann, "Deterministic Single-Agent Path Optimization in 2D", *IEEE 18th International Conference on Automation Science and Engineering (CASE)*, Oral presentation Paper, 2022

Maryam Khazaei, Carlos Alvarenga, Marcelo Kallmann,"Path Smoothing with Deterministic Shortcuts", 2022 IEEE Robotic Computing (IRC), page 411-415, Italy, 2022

Maryam Khazaei, Matthew Morozov, Marcelo Kallmann,"Optimizing Curvature and Clearance of Piecewise Bézier Paths", *IEEE International Conference on Control, Mechatronics and Automation (ICCMA)*, Norway, 2023

Maryam Khazaei, Matthew Morozov, Marcelo Kallmann,"Multi-Objective Multi-Robot Path Planning in Continuous Environment using Convex Optimization", *To be submitted*, 2023

Maryam Khazaei, Lori Lewis, "A Review on Higher-Order Spline Techniques for Solving Burgers Equation Using Variation of B-Spline Techniques", *International Conference on Advances in Applied and Engineering Mathematics*, San Francisco, United States, 2022

Maryam Khazaei, Yeganeh Karamipour, "The Spline Collocation Method for Solution of The Linear Seventh Order Boundary Value Problems", *Journal of Applied Mathematics and Physics*, page 3058-3066, 2021

Jalil Rashidinia, Maryam Khazaei, Hassan Nikmarvani, "Spline collocation method for solution of higher order linear boundary value problems", *Turkic World Mathematical Society (TWMS) Journal of Pure and Applied Mathematics*, page 38-47, 2015

ABSTRACT OF THE DISSERTATION

**Lane Determination and Optimization for Multi-Agent Systems**

by

Maryam Khazaei Pool
Doctor of Philosophy in Electrical Engineering and Computer Science
in the Graduate Division of
the University of California, Merced

University of California Merced, Fall 2023

Professor Sungjin Im, Chair

Path planning and path optimization are enduring areas of interest in the fields of computer science and automation. With the increasing presence of autonomous mobile robots in industries such as agriculture, manufacturing, and shipping, it is essential that we develop and improve methods for both planning and optimizing plans for agents in diverse environments. In this thesis I focus on path smoothing and optimization methods with customized properties and their application to optimizing multiple non-intersecting paths, which we also refer to as sets of lanes.

First I propose a new path smoothing method based on the approach of deterministic shortcuts. The presented method addresses limitations of the the popular random shortcuts method, and demonstrates superior results as measured both in terms of path length and smoothness. As common in such methods, the path is represented here as a polygonal line and smoothness quality is measured as the worst angle on the path. The proposed approach also provides user-specified quality-based termination conditions.

I then present a method that addresses a continuous path representation, by employing piecewise Bézier curves for the path representation. In this way, the optimized paths guarantee $C^1$ continuity and also enable taking into account curvature constraints, which are important on a number of applications. Given this

representation, I then propose a new path optimization scheme based on convex optimization of piecewise quadratic Bézier paths. The proposed method targets a minimum required clearance from obstacles while minimizing path length and maximum curvature. The user is able to customize the objective function by assigning different weights to the clearance, length, and curvature terms.

Finally I introduce a priority-based approach to improve the application of the piecewise quadratic Bézier optimization scheme to multi-agent applications where a set of non-intersecting paths, or lanes, needs to be optimized. The proposed approach is called Multi-agent Interleaving Convex Optimization (MICO), where a priority-based approach is used to interleave the optimization of the individual lanes, such that convergence is achieved faster than alternative approaches. I present results comparing MICO to two other optimization strategies: the first is a Multi-agent Baseline Convex Optimization (MBCO) approach where lane optimization is interleaved without any particular priority, and the second method, called Piecewise Quadratic Bézier Curve Multi-agent Convex Optimization (MCO), applies the same optimization scheme to all the piecewise Quadratic Bézier paths simultaneously in a single optimization scheme.

Users can still fine-tune the objective function by assigning different weights to clearance, length, and curvature terms, ensuring adaptability to various scenarios.

In conclusion, this work provides three new approaches to path optimization for multi-agent systems. The proposed methods address path optimization schemes that simultaneously optimizes for clearance, length, and curvature, providing flexibility and adaptability for various application requirements. This work also opens the door to the problem of developing multi-lane path planning and smoothing approaches, providing valuable tools for addressing certain types of multi-agent planning problems in robotics, autonomous vehicles, and simulated autonomous agents.

# Chapter 1

# Introduction

Path planning and path optimization are two important areas of research in the fields of computer science and automation. Path planning and optimization are crucial for the efficient and safe operation of autonomous mobile robots in various industries. In the manufacturing industry, path planning and optimization techniques can help to minimize the time taken by robots to move between different workstations, thereby increasing productivity and reducing costs. For instance, a recent study by Jin et al [47] proposed an optimized path planning technique for additive manufacturing that helps to improve the quality and uniformity of printed parts. Similarly, in the shipping industry, path planning and optimization can help to reduce fuel consumption and minimize the time taken by ships to reach their destinations. Modern route planning systems use global weather forecasts to build optimal routes, which can help to ensure that ships arrive safely and on time at their ports of destination [38, 46, 47]. These examples demonstrate the importance of path planning and optimization in diverse industries and the potential benefits that can be achieved by developing and improving these methods [70].

Path planning and optimization may be applied to 2-dimensional, 3-dimensional, or higher-dimensional configuration spaces. For path planning, the goal is generally to find a continuous set of configurations connecting an initial configuration to a goal configuration which satisfy certain environmental constraints, such as not crossing obstacles or limiting the range of motion of an actuator that will be used to execute the path. In 2 dimensions, we may represent a path as a set of piecewise

linear segments. Alternative path representations use higher order curves to better represent smooth executable paths. Path optimization seeks to improve a given path according to domain-specific or user-defined criteria. Updates may come in the form of changing a single vertex, path segment, or the entire representation of the path at once. Methods for both path planning and path optimization are discussed in greater detail in the literature review sections and in the state of the art discussion included in each chapter.

In my research, I have explored the topic of path optimization and proposed several novel methods, such as the Deterministic Shortcut-based Smoothing method, and Quadratic Bézier Convex Optimization methods for single-agent and multi-agent systems. In developing each method, I have considered the trade-offs of the existing methods in the literature and demonstrated how I was able to push forward the state of the art. This involved discovering the practical challenges and advantages when using these techniques in real-world situations, especially when dealing with multi-lane systems. Chapter 2 establishes context for this work in the broader literature. In this section, I discuss the existing optimization techniques for path smoothing such as Shortcut methods, Numerical optimization method, and Geometric optimization methods.

My first work proposed a shortcut-based path smoothing method which utilized a novel technique for determining the free space available to vertices of a linear path. Chapter 3 presents this method, which is named Deterministic Shortcut Smoothing (DSS). Here, a polygonal path is considered smooth if the angle between path segments at each vertex is large. We do not consider smoothness beyond this definition for this section. The goal of this definition of smoothness is to approximate a continuous differentiable path. We compare against the popular random shortcut technique and we show to improve some of its weaknesses, which include not having quality-based termination conditions, and being prone to miss smoothing tight areas which are unlikely to be sampled. As a result, our prioritized shortcut selection and quality-based termination conditions result in a method that outperforms the random shortcuts approach both in terms of path length, average angle, worst-case angle, and running time. Polygonal paths, which

are comprised of linear segments, are not suitable for all applications. My main interest in developing this method was to apply it to optimizing and improving the quality of multiple lanes in a lane system. Therefore, my next chapter addresses a path representation relying on Bézier curves. This way, I was able to address a continuous definition of smoothness, such as $C^1$ continuity.

My second work described in chapter 4 presents a novel method for path optimization that combines piecewise quadratic Bézier curves with convex optimization to produce smooth, and obstacle-avoiding paths. Starting with a transformation method that turns piecewise linear paths into piecewise quadratic Bézier curves, we discuss the challenges of ensuring $C1$ continuity and shaping the objective function to balance the trade-offs between path length, curvature, and being collision-free. The approach, referred to as the Quadratic Bézier Convex Optimization (QBCO) method, prioritizes not only the length of the path but also takes into account crucial aspects such as continuity, curvature, and obstacle clearance.

A distinctive feature of QBCO lies in its user-friendliness, allowing individuals to customize the objective function by assigning specific weights to clearance, length, and curvature parameters. As a result, QBCO emerges as a versatile solution with promising applications in robotics and autonomous vehicles. QBCO's distinguishing feature lies in its adaptability, empowering users to fine-tune the objective function of our convex optimization problem to their specific requirements.

Our previously proposed work considered path optimization for only one agent at a time and used a standard Rapidly-Exploring Random Tree (RRT) method [1, 3] to generate the paths used to test the presented optimization methods. For the work proposed in chapter 5, I have explored a rich set of new problems from expanding the scope of planning to generating multiple non-intersecting paths, or lanes, and then optimizing the lane systems for multiple agents. In chapter 5, I propose three convex optimization algorithms, each of which is tested by first generating paths with a multi-agent version of RRT, refining them through multi-agent DSS, and transforming piecewise linear paths into piecewise quadratic Bézier curves for further smoothing while maintaining $C^1$ continuity. This method maintains the user-specified objective function from my work in the single-agent

case, while solving challenges unique to multi-agent systems. In this work, I do not focus on developing a new planner. Instead, I modified existing planners to generate multi-lane systems that can be optimized using the path optimization methods presented in this work.

The Multi-agent Baseline Convex Optimization Method (MBCO) adopts a sequential and individual optimization strategy for each path lane. This straightforward approach optimizes one lane at a time, treating each lane as a separate entity. The optimization process utilizes single-agent optimization technique discussed chapter 4, considering each other lane as an obstacle for the current lane. The algorithm iteratively adjusts paths using convex optimization, accounting for real obstacles, and returns an efficient set of paths.

Our proposed algorithm, Multi-agent Interleaving Convex Optimization Method (MICO), presents a novel approach by iteratively optimizing each lane in turns. It initiates with a small optimization step for lane 1, progresses to lane 2, and repeats the process. Similar to the MBCO, it employs a single-agent optimization technique, treating other lanes as obstacles. The prioritization criteria are heuristic-based, considering factors such as path length, the sharpest angle of the path, and proximity to obstacles to determine the order of optimization. The algorithm uses a weighted scoring system to select the path with the highest overall score for optimization, contributing to a faster convergence time for all paths.

We refer to the global convex optimization of all paths in the multi-lane system as the Piecewise Quadratic Bézier Curve Multi-agent Convex Optimization (MCO). This method creates a single convex optimization problem for all the paths together, minimizing a combination of factors such as curvature, clearance, and length. The problem is subject to various constraints, ensuring continuity and specific conditions at the start, end, and critical points of the path. The algorithm runs the optimization program iteratively, updating the Bézier path with the values calculated through convex optimization until convergence is reached. This iterative process allows for capturing changes in the path's shape and updating distance constraints for precise optimization, providing an improved set of paths for multi-agent path planning.

**Main Contributions**: In summary, this thesis presents contributions to the research community in the areas of single and multi-path optimization. The proposed methods address both single and multi-agent systems in 2D environments with diverse obstacle configurations. The contributions include a deterministic shortcut method for polygonal paths, a convex optimization model for paths defined as piecewise Bézier curves, and an optimized priority-base approach to address systems of multiple lanes for multi-agent problems. Furthermore, I extended the classical RRT path planner to be able to plan multiple non-intersecting paths simultaneously with high probability of success in order to generate multi-lane solutions that were used in the presented benchmarks and evaluations.

# Chapter 2

# Literature Review

Path planning and optimization techniques have been extensively researched in the fields of computer science and automation.

This literature review briefly discusses popular approaches for path planning and discusses optimization techniques based on shortcuts, geometry, and numerical solvers. We highlight the specific case of trajectory optimization and the constraints that systems with dynamic models present. Finally we summarize the approaches as they apply to multi-agent or multi-lane systems, as these are the most relevant for our most recent works.

## 2.1 Sampling-based Path Planners

Sampling-based path planners are a family of algorithms that are used to solve path planning problems. These planners create possible paths by randomly adding points to a graph or tree until some solution is found or time expires. They are probabilistic complete, meaning that the probability to find a path approaches one when time goes to infinity [33,66]. PRMs and RRTs are two of the most commonly used path planning algorithms, with many variations available. Additionally, there are asymptotically optimal versions of these algorithms, namely PRM* and RRT*. PRM* and RRT* are extensions of PRMs and RRTs, respectively, that are designed to find the optimal solution to the path planning problem. They are shown to be asymptotically optimal, with the probability of finding the optimal solution

approaching unity as the number of iterations approaches infinity. Two prominent examples of sampling-based planners are Rapidly-exploring Random Trees (RRT) and Probabilistic Roadmaps (PRM). RRT can be understood as growing a single tree from a robot's starting point until one of its branches hits a goal. PRM create a tree by randomly sampling points in the state-space, testing whether they are collision-free, connecting them with neighboring points using paths that reflect the kinematics of a robot, and then using classical graph shortest path algorithms to find shortest paths on the resulting structure. These algorithms have been used in a variety of applications, including additive manufacturing and mobile robot path planning [33, 51, 66, 74].

## Rapidly-Exploring Random Tree Planner

Rapidly-exploring Random Tree (RRT) methods, have attracted attention over the last decade and are proposed as effective planning algorithms in high-dimensional spaces [1, 3, 58, 101].

The Rapidly-exploring Random Tree (RRT) algorithm is a motion planning algorithm that efficiently searches non-convex, N-dimensional spaces by randomly building a space-filling tree. The algorithm was developed by Steven M. LaValle and James J. Kuffner Jr.

The idea of the algorithm for RRT is explained as follows:

- Initialize the tree $T$ with the initial configuration $q_{\text{init}}$.
- Repeat the following steps until either a goal configuration $q_{\text{goal}}$ is added to $T$ or a time limit is reached:
- Generate a random configuration $q_{\text{rand}}$.
- Find the node $n$ in $T$ that is closest to $q_{\text{rand}}$.
- Steer from node $n$ towards $q_{\text{rand}}$ to generate a new configuration $q_{\text{new}}$. If the path from node $n$ to $q_{\text{new}}$ is collision-free, we add $q_{\text{new}}$ to $T$ and add an edge from node $n$ to $q_{\text{new}}$.

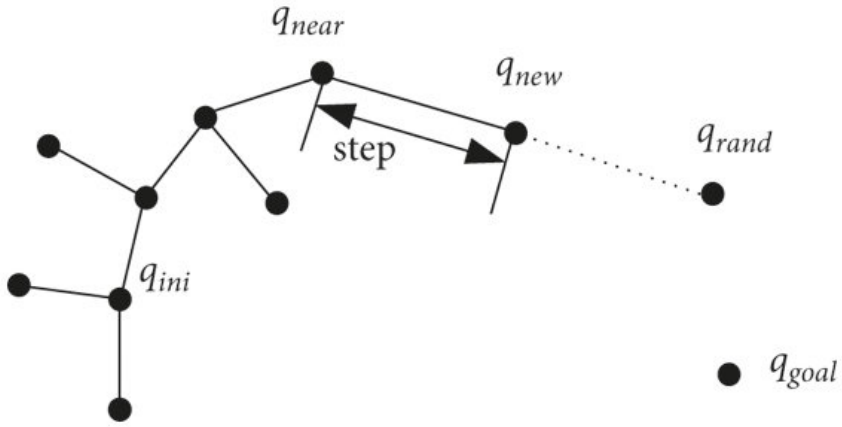


Figure 2.1: This image represents a visualization of the RRT algorithm's path planning process.

In the visual representation of Figure 2.1, we can see the tree growing from the initial configuration $q_{\text{init}}$ towards the goal configuration $q_{\text{goal}}$. The algorithm explores the configuration space by repeatedly extending the tree towards random points $q_{\text{rand}}$, ultimately finding a path from the start to the goal (if one exists) while avoiding obstacles. The dynamic nature of the RRT algorithm is showcased as it efficiently explores and expands towards the goal. Sampling-based algorithms depend on collision checking in order to determine the feasibility of possible trajectories. This sampling generates a graphic road-map which avoids an explicit representation of the Configuration Space. These algorithms provide probabilistic completeness. This means that, as more samples are taken, the probability that the planner fails to find a path asymptotically approaches zero, if a path exists. Piecewise-linear trajectories obtained from RRT may be jerky or have sharp turns, which effects the efficiency of agents following those trajectories. As a result, the importance of generating smooth trajectories and feasible motions is considered. Ravankar et al [86] summarized the state of the art, noting the importance across all smoothing methods of considering dynamic and kinematic constraints in path planning and optimization. Dynamic properties of robots also play a major role in control theory, as exemplified by a paper by Wu et al [95] on the topic of controlling

5-DOF machine tools.

Path smoothing and optimization is a topic which has been studied by many researchers. Path optimization techniques play a crucial role in refining the generated paths to improve their quality. Researchers have explored different optimization objectives, such as minimizing path length, reducing energy consumption, and optimizing other performance metrics [23]. Trajectory optimization builds upon path optimization by considering the dynamics and constraints of the robot. It aims to find an optimal control solution that guides the robot along the desired path while satisfying various constraints, such as actuator limits and obstacle avoidance. Optimal control techniques, such as model predictive control (MPC) and direct collocation methods, have been extensively studied for trajectory optimization. While some methods, like our discrete shortcut-based smoothing method [56], are based on iterative shortcuts, a wide range of techniques focusing on the generation of smooth paths have been proposed. From a broad view, one possible classification for single-robot path planning is outlined by numerical optimization-based methods, geometric-based methods and shortcut-based iterative methods.

## Probabilistic Roadmap Planner

The probabilistic roadmap planner (PRM) has been studied by many researchers [2, 12]. The probabilistic roadmap planner (PRM) is a planning algorithm in robotics that helps robots find a path between a starting configuration and a goal configuration while avoiding collisions. The PRM takes random samples from the robot's configuration space, testing them to see if they are in free space, and uses a local planner to connect these configurations to other nearby configurations. The starting and goal configurations are added to the graph, and a graph search algorithm is applied to determine a path between them.

Comprising two main phases, PRM begins with a construction phase wherein a roadmap or graph is created to approximate feasible motions within the environment. This involves the generation of random configurations that are subsequently connected to neighbors, typically defined by either the k nearest neighbors or all neighbors within a predetermined distance. Configurations and connections are

continuously added to the graph until a satisfactory level of density is achieved. In the subsequent query phase, the start and goal configurations are linked to the graph, and the path is derived through a Dijkstra's shortest path query.

Given certain relatively weak conditions on the shape of the free space, PRM is probabilistically complete. The rate of convergence depends on certain visibility properties of the free space, where visibility is determined by the local planner. If each point can see a large fraction of the space, and also if a large fraction of each subset of the space can see a large fraction of its complement, then the planner will find a path quickly. The invention of the PRM method is credited to Lydia E. Kavraki [52, 53, 94]. There are many variants on the basic PRM method, some quite sophisticated, that vary the sampling strategy and connection strategy to achieve faster performance [81].

## 2.2 Shortest-Path Algorithms

A shortest-path algorithm is a method to find the path between two vertices in a graph that has the minimum cost. The shortest-path problem is a well-studied topic in computer science, particularly in graph theory. An optimal shortest-path is one that satisfies the minimum length criteria from a source to a destination. Due to the problem's numerous and diverse applications, there has been a surge of research in shortest-path algorithms. These applications include network routing protocols, route planning, traffic control, path finding in social networks, computer games, and transportation systems, among others [71].

Most shortest-path algorithms can be grouped into two main types. The first type focuses on finding the shortest paths from one starting point to all other points in the graph, known as single source shortest-path (SSSP). The second type determines the shortest paths between every pair of vertices in the graph, which is called all-pairs shortest-path (APSP).

The classification system illustrated in the subsequent Figure 2.2 provides a taxonomy for organizing various categories of shortest-path problems. This is one approach to structuring such a classification [71].

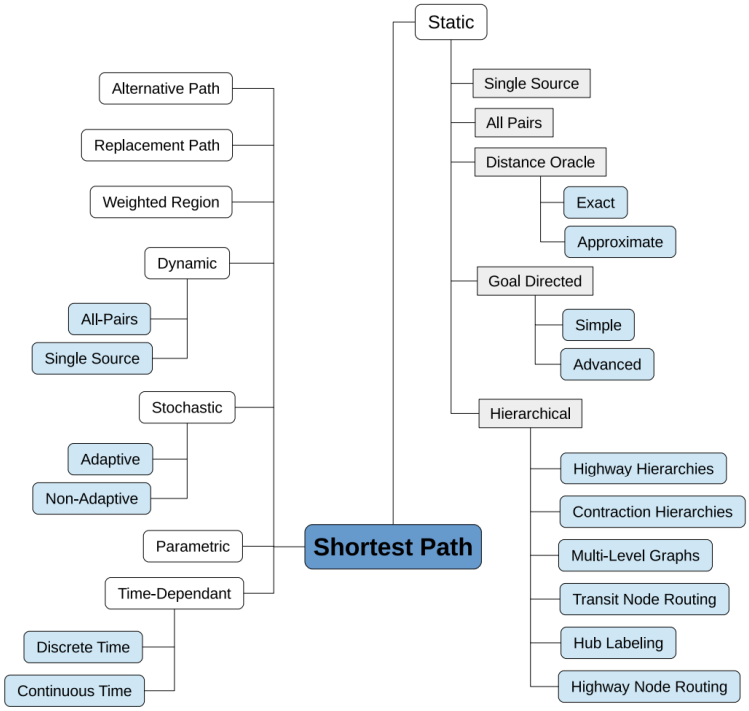


Figure 2.2: Taxonomy of Shortest-Path Algorithms

The static branch in Figure 2.2 lists algorithms that operate over graphs with fixed weights for each edge. The weights can represent distance, travel time, cost, or any other weighting criteria. The SSSP algorithms compute the shortest-path from a given vertex to all other vertices. The APSP algorithms compute the shortest-paths between all pairs of vertices in the graph. Hierarchical algorithms break the shortest-path problem into a linear complexity problem, which can lead to enhanced performance in computation by orders of magnitude. Goal-directed algorithms optimize in terms of distance or time toward the target solution. Distance oracle algorithms include a pre-processing step to speed up the shortest-path query time. Distance oracle algorithms can either be exact or approximate. The dynamic branch in Figure 2.2 lists algorithms that process update or query operations on a graph over time [71]. The update operation can insert or delete edges from the graph, or update the edge weights. The query operation computes the distance between source and destination vertices. Dynamic algorithms include

both APSP and SSSP algorithms. Time-dependent algorithms target graphs that change over time in a predictable fashion. Stochastic shortest-path algorithms capture the uncertainty associated with the edges by modeling them as random variables. Parametric shortest-path algorithms compute a solution based on all values of a specific parameter. Replacement path algorithms compute a solution that avoids a specified edge, for every edge between the source vertex and the destination vertex. Replacement paths algorithms achieve good performance by reusing the computations of each edge it avoids. On the other hand, alternative path algorithms also compute a shortest path between vertices that avoids a specified edge [71]. The distinguishing factor between both categories is that replacement path algorithms compute a solution for every edge between the source vertex and the destination vertex, whereas alternative path algorithms only compute a single shortest path between vertices that avoids a specified edge [71].

Practical surveys such as [105] have been conducted on the topic of shortest-path algorithms. This survey focuses on exact and approximate shortest-path algorithms, including single-source shortest-path (SSSP), all pairs shortest-path (APSP), spanners, and distance oracles. The survey highlights the various variations that each category adopts when handling negative and non-negative edge weights as well as directed and undirected graphs [105].

Algorithms intended for traffic applications, particularly route planning techniques, are the subject of numerous surveys. Holzer et al [42] categorize Dijkstra's algorithm variants based on the speedup techniques that are used. The kind of data used in oracle methods greatly influences how effective speed-up strategies are. Furthermore, the optimal speedup method is contingent upon the RAM, layout, and acceptable preparation time. The goal of heuristic algorithms is to reduce calculation time. The primary characteristics of heuristic algorithms and their computational expenses are suggested by the survey [31].

Good worst-case and average-case bounds over a graph are demonstrated by the survey [36]. From a theoretical perspective, Goldberg [36] looks into how well point-to-point shortest path algorithms perform over road networks. In addition to reviewing algorithms like Dijkstra and $A*$, Goldberg provides examples of heuristic

methods for calculating the shortest path given a section of the graph.

Farias et al [30] present an improved method for computing shortest path maps using OpenGL shaders. The authors use GPU rasterization to propagate optimal costs on a 2D environment, producing efficient shortest path maps. The method handles both point and line segment sources. The paths produced have global optimality, which is often neglected in animated virtual environments. The approach is suitable for animating multiple agents moving toward entrances or exits in a virtual environment. This work focuses on the Multi-agent Random Shortcut Method, where initial paths are derived from the Max Flow algorithm. In contrast, our research emphasizes Multi-agent Convex Optimization on piece-wise Bezier curves, with initial paths obtained from the Rapidly-exploring Random Tree (RRT) algorithm. This highlights the different approaches and methodologies employed in path optimization. While Random shortcut method can be simpler and faster for some problems due to its straightforward implementation and the fact that it does not require complex computations or the need to satisfy any constraints [8], piecewise Bezier curves, compared to piecewise linear paths, offer smoothness, flexibility in shape manipulation, and curvature control. They provide a better physical interpretation of motion, making them suitable for applications like computer graphics and autonomous vehicle path planning. Moreover, the proposed convex optimization methods have C1 continuity and allow for the incorporation of constraints, making it a more flexible and powerful tool for path optimization. The Max Flow algorithm is designed to identify the maximum number of paths traversing from a start region to a goal region. On the other hand, the work on this thesis focus on the optimization phase. The Rapidly-exploring Random Tree (RRT) algorithm used to compute multiple lanes does not guarantee finding the maximum number of paths however it is extendible to multiple dimensions, which is one of the reasons for the popularity of such methods.

## 2.3 Numerical Path Optimization Techniques

In terms of trajectories, optimization techniques require appropriate objective functions, collision checks with obstacles, and constraints on velocity and acceleration in order to build smooth trajectories [20]. Studies show that applying optimization techniques such as B-spline trajectory planning have resulted in high-quality trajectories in terms of smoothness and in terms of reducing unnecessary motions [20]. One of the limitations of these methods is the reliance on an initial trajectory. These initial trajectories are relatively simple, and the proposed optimization methods are not effective on complicated trajectories.

A Chaos-based Particle Swarm Optimization (CPSO) system has been proposed to find control points for a Bézier curve so as to produce a path that avoids collisions and minimizes path length [91]. The authors studied how different chaos maps effect the final solution and demonstrated that the proposed method outperforms non-chaos PSO. The algorithm performs local spline refinement to compute smooth, collision-free paths in narrow passages and satisfy velocity and acceleration constraints.

Several smoothing techniques relying on curve interpolation such as Bézier Curves [90], Splines [59] and Polynomial Basis Functions [84] have been developed. Considering the fact that B-spline curves take local modifications into consideration without the overall path changes, B-spline curves are used for describing the motion trajectory of robot [78]. The problem of shape representation using free-form curves or surfaces is solved with B-spline curves in computer graphics [55, 73].

Pan et al [80] presented a path optimization technique include a local spline refinement to compute smooth, collision-free paths in narrow passages and satisfy velocity and acceleration constraints.

CHOMP optimizes an initial trajectory iteratively using functional gradient techniques [104]. Despite some techniques focusing on computing feasible paths, CHOMP optimizes in dynamic environments and uses task-based criteria. Using Covariant Hamiltonian gradient descent, CHOMP minimizes trajectory velocities while keeping configurations collision-free for the agents. However, finding the

global optimum might lead to obtaining high-cost local minima. Additionally, CHOMP does not function properly when the initial trajectory is far from obstacles.

Stochastic Trajectory Optimization for Motion Planning (STOMP) is an algorithm introduced by Kalakrisshnan et al [49] inspired by the CHOMP algorithm. This algorithm outperforms gradient based-methods like CHOMP when finding the local minimum cost function. STOMP samples a series of noisy trajectories near to the initial path to reduce trajectory cost. One of the limitations of STOMP and CHOMP is their performance when finding feasible solutions when the number of constraints is high.

Heiden et al [40] proposed a Gradient-informed post-smoothing algorithm with two phases: deformation of trajectories by the placement of vertices; and pruning of the path with shortcuts. These phases generate continuous trajectories which can avoid collision in dynamic environments.

Yang et al [97] presented a gradient-free optimization technique which they call the Double Layer Ant Algorithm. The authors perform Turning Point Optimization and Piecewise B-spline smoothing to improve the initial path.

## 2.4 Geometric Path Optimization Techniques

Optimizers that rely on geometric path representations are quite popular. For example, Choi and colleagues [16–19] have developed path planning algorithms for autonomous vehicles that use Bézier curves to generate paths, taking into account waypoint and corridor constraints.

Similar to our work, they break the path into segments using Bézier curves and aim to make the path smooth while ensuring it does not have sharp turns. They also ensure the path is collision-free by dividing the space into sections within a specified corridor. However, their approach requires pre-computing the corridor and does not allow for customization based on factors like curvature and clearance from obstacles.

In contrast, our approach automatically defines a corridor using a set of free

disks placed around the path. This corridor adapts as the optimization process unfolds. Unlike the approach by Choi et al [17], we do not restrict the control points of the Bézier curve to be inside the free space. Instead, we allow control points to extend outside the free space as long as the path remains collision-free. Additionally, our approach lets users customize the importance of factors like path length, curvature, and clearance in the optimization process.

Another related work is by Cimurs et al [21], where they use Bézier curves for path smoothing. However, their method focuses on simplifying paths and ensuring they are short and smooth without using optimization techniques. They also do not consider curvature as a part of the smoothing process.

## 2.5 Shortcut-based Optimization Techniques

A lack of simple and effective methods with quality-based termination conditions can be observed in the traditional shortcut methods [35, 39, 43, 82]. Random shortcut heuristic methods replace jerky portions of a path with shorter segments in the configuration space and check if they are collision-free. The sub-path between the two vertices on the path will be replaced with a straight segment if it is collision free. The implementation of shortcut techniques is simple, fast, and produces high-quality paths in many cases [35, 50, 54]. However, these implementations might not be able to provide higher-order smoothness. Also, collision checking along higher order trajectories is costly and the random selection of shortcuts may miss tight areas difficult to sample. This may lead to sharp corners in tight areas, making it difficult to achieve termination conditions based on path quality. Traditional shortcut methods operate as 'anytime' functions, without natural stopping conditions, and may be ineffective at optimizing smaller sections of the path. Thus, random shortcut methods are not efficient in cases where only a small portion of the path needs to be optimized [8].

## 2.6 Trajectory Optimization Techniques

This thesis primarily deals with static path optimization, but there is also considerable work being done optimizing and planning for agents where dynamic conditions and factors must be considered. Specializing around dynamic factors is beyond the scope of the algorithms presented in this thesis, but such constraints can be straightforwardly included in future iterations of the proposed algorithms. Nevertheless, we provide an overview of optimization techniques focusing on dynamically-modelled trajectories here.

Numerous studies have demonstrated the crucial role of trajectory design in ensuring the stability and enhanced control of dynamic systems [92]. The process of creating a trajectory that satisfies a set of restrictions and minimizes (or optimizes) a performance metric is known as trajectory optimization. Trajectory optimization, in general, refers to a method for calculating an open-loop solution to an optimal control issue. It is frequently applied to systems for which it is impractical, impossible, or not necessary to compute the whole closed-loop solution. The direct multiple shooting strategy is a conventional technique for direct methods that has been applied to real-world issues [9, 22]. Global collocation approaches have received a lot of interest lately, and a lot of work has been done in this area [48]. For example, Fahroo and Ross [29] developed a Chebyshev pseudospectral method to solve Bolza trajectory optimization problems in general that involve control and state constraints. All direct approaches, however, seek to translate the difficulties of continuous-time optimum control into nonlinear programming problems (NLP) [9, 68, 79].

Well-developed optimization techniques can solve the resultant NLP numerically. Using the trajectory optimization problem, several path planning techniques have been developed during the past ten years for both aerial and surface vehicles. Four conditions must be satisfied for a trajectory planning algorithm to be effective. First, the motion planning approach must always be able to determine the optimal path in real static scenarios. It must also be flexible enough to adjust to shifting circumstances. Thirdly, it ought to support and reinforce the chosen self-referencing technique. Fourth, it needs to minimize computing time, data storage,

and complexity [75]. The most popular optimization methods among these two groups are summarized and tabulated in [9].

The motivation behind employing dynamic programming-based approaches stems from their improved capability to achieve consistent performance and address local optimal solutions inherent in nonlinear optimal control problems.

In the overview of trajectory optimization techniques, several popular deterministic optimization algorithms are highlighted for their applicability to trajectory optimization problems. These include Sequential Quadratic Programming [41], the Interior-Point Method [60], Interior-Point Sequential Quadratic Programming [10], Linear Programming [67], Second-Order Cone Programming, Semidefinite Programming, Dynamic Programming, Differential Dynamic Programming, and Stochastic Differential Dynamic Programming. Each of these algorithms offers unique advantages in solving specific types of optimization problems, making them valuable tools in the field of trajectory optimization [9].

## 2.7 Multi-agent Path Optimization Techniques

In the area of multi-agent path optimization, the exploration of shortcut-based approaches appears relatively limited. Considering the various methods and conceptual variations in how these algorithms function and navigate multiple agents through their paths, multi-agent path-planning algorithms can be categorized into geometry-based methods and numerical optimization methods.

Wang et al [93] propose a method that uses topological reasoning to assign different paths to each agent, so that they can avoid congestion and reduce their travel time. The paper also presents a fast re-planning algorithm and a potential field based controller that enables robots to avoid collisions with each other while following their assigned paths.

Alotaibi et al [4] present a survey of sampling-based algorithms in their paper, including the methods used to solve the problem of finding optimal paths for multiple autonomous vehicles in a city environment. For a centralized approach, multi-agent path-planning RRT outperforms push-and-rotate, push-and-swap, and

the Bibox algorithm in optimizing solutions and navigating the search space within an urban environment.

To enhance remote sensing and expand coverage using multiple agents, Avellar et al [6] present a method that models the task as a graph and solves a mixed integer linear programming problem. The paper states that their method can find the optimal number of UAVs to use, depending on the area and the vehicles. Cho et al [14] present a method for planning the optimal path for a group of UAVs to cover a ground area with aerial images in a maritime search and rescue scenario. The method first divides the area into hexagonal cells and assigns a node to each cell center; and then it solves a mixed integer linear programming problem to find the shortest path for each UAV to visit all the nodes, considering the constraints of the UAVs' performance and the setup time.

Wu et al [96] extended this work to the problem of path planning for multiple agents in a dynamic environment. The authors of this paper propose an algorithm that combines an improved artificial potential field (APF) method and a B-spline curve optimization technique. The improved APF method introduces a gain constraint and a random factor to reduce the path oscillation and avoid the local minimum problem that affects the traditional APF method.

While similar to our work, they use optimization techniques for multi-agents to refine planned paths and reduce path curvature, our contribution lies in a customizable objective function that allows users to influence clearance terms. We optimize for quadratic Bézier curves, prioritizing simplicity in mathematical representation, computational efficiency, and space optimization over the cubic B-spline curves employed by the previous work.

Heuristic algorithms are particularly designed to prioritize efficiency when finding optimal paths. Within this category, two specific algorithms, $A*$ and $D*$ Search, are highlighted as prime examples of effective and streamlined approaches to path planning. Lurz et al [69] presents a method for path planning and reconfiguration for rigid multi-agent formations using splines. The authors describe a method of path planning that combines the benefits of the relaxed-$A*$ algorithm for quick initial path-finding with the smooth and controlled path optimization

provided by Bézier curves, splines, including considerations for restricting acceleration and velocity.

Yuan et al [99] propose a path planning algorithm which aims to avoid collisions and conflicts among multiple automated guided vehicles (AGVs) in complex environments. The algorithm utilizes an enhanced $A*$ and dynamic RRT to plan and generate paths for AGVs. Addressing the multi-AGV routing challenge, the improved $A*$ algorithm strategically devises a global path. Concurrently, the dynamic RRT algorithm is applied to establish a viable local path that conforms to kinematic constraints, thereby effectively averting collisions within the grid map.

The $D*$ algorithm is utilized for collaborative navigation among multiple robots through a knowledge-sharing mechanism facilitated by sensors by Ravankar et al [85]. The $D*$ algorithm enables robots to communicate critical environmental updates, such as the presence of new static obstacles or path blockages, and can be expanded for real-time applications in mobile scenarios.

Artificial Intelligence (AI)-based approaches lead the way in multi-agent path planning, leveraging cutting-edge technology to handle challenges in dynamic environments. The common subcategories involve a range of techniques, such as machine learning, optimization, and utilizing the capabilities of AI for multi-agent systems. The emphasis on AI arises from its extensive application and flexibility in tackling the evolving complexities of multi-robot scenarios, making it a key driver of innovation in intelligent path planning.

Zohdi et al [103] develops and tests a method for multiple drones to work together and map complex areas using machine-learning. This work introduces a machine-learning method for UAVs to plan their paths. The UAVs use reinforcement learning to learn from their actions and outcomes. The paper sets a reward function based on information, energy, and safety. It also suggests a heuristic algorithm that adjusts to different situations. By analyzing the image data to identify the exact situation in the environment, the convolutional neural networks enable robot navigation with a novel multi-robot path-planning algorithm that uses Deep Q-learning [7].

Deits et al [24] developed a method for guiding UAVs using mixed-integer op-

timization, but it's computationally intensive due to the division of environments into convex shapes. Our approach simplifies this, avoiding environment division, and is less time-consuming. It addresses their method's limitations in computational complexity, adaptability, and path refinement. Their work lacks emphasis on clearance and curvature control, and restricts paths to single safe regions, limiting global optimality claims. It also relies on specific convex regions, requiring intelligent seed point selection in complex environments. They face challenges in ensuring smooth control inputs and solving problems without numerical difficulties.

# Chapter 3

# Path Smoothing by Deterministic Shortcuts

## 3.1 Introduction

Path smoothing is an important operation that appears in a number of path planning applications. Path smoothing is often used to smooth the result of a sampling-based planner, or to deform a path in order to achieve desired qualities, such as maintaining a desired distance from obstacles or controlling a given quality aspect. In applications relying on multiple paths, such as in multi-agent path finding problems, relying on a simple and efficient smoothing method with controlled quality becomes particularly important given that the quality of one smoothed path may influence the space available for smoothing the other paths.

In this chapter, we propose a Deterministic Shortcut-based Smoothing (DSS) method which overcomes the main limitations of previous shortcut-based methods by being able to consider user-specified termination conditions based on solution quality. At each iteration, our method first identifies a vertex on the path that has the most potential for path improvement, and then applies one of two possible shortcut-based smoothing operations.

As a result, our prioritized shortcut selection and quality-based termination conditions result in a method that outperforms a traditional implementation of

the random shortcut approach, both in terms of path length and in worst-case angle measured along the path. In this work, similarly to previous work on this area, we consider a path to be represented as a polygonal line. We present several benchmarks demonstrating that, for the same amount of smoothing time, our method produces higher-quality paths when compared to the traditional random shortcut approach [35, 39, 43, 82].

## 3.2 Related Work

The method proposed in this work is most related to methods based on random shortcuts, which represent a popular approach that is simple and effective [35, 39, 43,82]. However, without extensions, random shortcut selection may fail to smooth a path in tight areas which are difficult to be sampled. This may lead to sharp corners in tight areas making it difficult to achieve termination conditions based on path quality.

Hsu et al [43] describe a short-cutting technique that removes redundant motions on path. The optimization approach used recursively breaks the path into two sub-paths and check whether the sub paths can be replaced by straight-line paths. Instead of only removing redundant vertices, our Deterministic Shortcut-based Smoothing (DSS) Algorithm takes nearby obstacles into account by using the Corner test described in Section 3.4 that lets the paths remain sufficiently far away from obstacles. Also, the termination condition in our DSS Algorithm is based on two criteria: angle and clearance. To measure the angle or smoothness of the current solution, we compute the sharpest angle in the polygonal path and test if it is under a desired threshold. Related previous work does not consider the angle at each vertex at all and therefore may result in paths which are impractical for a robot to follow. Further, in general, the conditions considered in Hsu's work are not as robust as ours and may leave large distances between the path and obstacles (larger than required clearance), which is often sub-optimal. The termination criteria in Hsu's work only consider successive iterations to improve the path and terminate if the improvement falls below some threshold. The work

of Hsu et al [43] also considers potential shortcuts in an order which is not robust because the technique described in Hsu's work considers the longest shortcuts first. However, this issue is solved by the angle criterion of the termination condition in our approach.

The Cutting-triangle's-edge algorithm presented by Guernane et al [37] produces shortcuts by connecting the midpoints of path segments for every adjacent edge in the path. The dynamic and kinematic constraints of the robot are used to define cubic polynomials which smooth edge discontinuities. One of the drawbacks of this technique is that there might not be the chance to create a shortcut for every corner of the path. In contrast, our method will try to make the best shortcut that it can for every vertex of the path, in a prioritized manner. Our method generates shortcuts that can cut out a larger portion of unneeded path by calculating the most free space in the environment (discussed in 3.4). Therefore, we believe our proposed method is able to create shorter paths than that of [37].

Another shortcut-based method was proposed by Campana et al [8], which uses backtracking when a collision is detected on the most recent iteration of the algorithm. However, while their work is based on the computation of a gradient, our method follows a simpler approach. We select vertices according to their potential for improving the path nearby it, using their distance to obstacles. Therefore our method does not need to perform an explicit collision check. Further, the method proposed by Campana does not explicitly take into consideration the angles formed at vertices along the path.

## 3.3 Definition and Models

In this section, we introduce notations and present the definitions used in the computation of the path. The definitions provided here match closely to and are largely based on those provided in *Planning Algorithms*, by Lavalle [62].

**Configuration Space (C-space)**

A Configuration Space (C-space) is the space of all possible transformations that could be applied to a robot. Thus, each discrete transformation or set of pa-

rameters defining robot pose, shape, and orientation may be called a configuration in the C-space.

$C_{free}$ and $C_{obstacle}$ are two regions within the C-space. $C_{obstacle}$ is the set of all configurations of the robot which bring it into collision with one or more obstacles in the environment. Thus $C_{free}$ is the collision-free configuration space $C_{free} = C \setminus C_{obstacle}$. We typically designate a configuration, $c_I \in C_{free}$ as the initial state with a configuration $c_T \in C_{free}$ designated as the target state. For simplicity, we denote initial state and the target state with $\mathbf{s}$, and $\mathbf{t}$ and call them the start and target points respectively.

**Single Agent Planning Problem** In this work, we consider the environment for a single agent planning problem to be set of obstacles denoted by $O$, which contain a polygonal boundary. The environment contains a Start point $\mathbf{s}$ and a Target point $\mathbf{t}$ between which our goal is to generate and optimize a path.

**Path** A path is defined as a continuous function $\Pi : [0, 1] \Rightarrow C$ with $\Pi(0) \in C_{\mathbf{s}}$ and $\Pi(1) \in C_{\mathbf{t}}$. $C_{\mathbf{s}}$ is the set of configurations which correspond to the Start point $s$. Likewise, $C_{\mathbf{t}}$ corresponds to $t$. For the path to be collision-free, its span must lie in $C_{free} \subseteq C$.

**Obstacle** An obstacle is a connected subset of the configuration space. When a robot occupies a configuration in this subspace, it is considered in collision with this obstacle. $C_{obstacle}$ may consist of many obstacles.

Our environment contains only stationary obstacles with known polygonal geometry. When planning, we avoid these obstacles in order to create and improve a collision-free path using $\mathbf{s}$ and $\mathbf{t}$. Obstacles may be convex or non-convex polygons in the environment.

**Agent** An agent is an entity capable of making decisions or following instructions. In this chapter, we are particularly interested in autonomous mobile robots acting as agents, and therefore we sometimes use the terms interchangeably.

**Clearance of a configuration** In this chapter, we call the minimum Euclidean distance from a configuration $c \in C$ to any obstacle the clearance of that configuration $c$.

Finding the clearance of every point along a path in this way is important in

making real-world collision-free plans. By keeping path clearance above a threshold, the path will not cause a robot to collide with an obstacle when attempting to execute said path in the real world because of the robot's finite size. For this reason, we use a definition of clearance which emphasizes the *minimum* distance to any obstacle, rather than the total volume around a configuration.

## 3.4 Methodology

In the scope of this work we address the particular case of 2D polygonal paths. We consider that the input polygonal path $P$ is defined by an ordered set $P = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_N\}$ containing $N$ vertices. The path connects the starting location $\mathbf{v}_1$ to the goal location $\mathbf{v}_N$. Environments are 2-dimensional and described by a set of polygonal obstacles $O = \{O_1, O_2, ..., O_M\}$.

Our goal is to minimize the length of $P$ and maximize the smallest angle between two adjacent $P$ segments, while maintaining $P$ collision-free and fixed at $\mathbf{v}_1$ and $\mathbf{v}_N$.

Our proposed Deterministic Shortcut-based Smoothing (DSS) method is based on two geometric shortcut determination procedures: the *Disk Test* and the *Corner Test*. These procedures will first test if a shortcut can be performed with respect to a given vertex, and if so, that shortcut is returned and the path is improved; otherwise, a label *done* is returned and the overall algorithm stops.

### 3.4.1 Disk Test

The Disk Test is outlined in algorithm 1. The Disk Test follows a greedy selection process: at every iteration, we optimize the path at the vertex with the most free space around it. First, for every non-terminal vertex $\mathbf{v}_i$, we start by calculating the minimum Euclidean distance to every obstacle in the environment:

$$d_{min}(\mathbf{v}_i) = \min_{O_j \in O} D(\mathbf{v}_i, O_j), \quad \forall i \in \{2, 3, ..., N-1\},$$

where $O$ is the set of obstacles, $O_j$ is an obstacle in the environment and $D$ is a function that returns the minimum distance from the polygonal obstacles to

a vertex in the path. This distance serves as a proxy for the free space available around a vertex. We then determine the vertex with the most free space $\mathbf{v}^*$, which is the vertex such that:

$$d_{min}(\mathbf{v}^*) = d^* = \max d_{min}(\mathbf{v}_i), \forall i \in \{2, 3, ..., N-1\}.$$

Distance $d^*$ is then used as the radius of a circle centered at $\mathbf{v}^*$, which we call $C$. The points of intersection between $C$ and the path form the endpoints of the shortcut with which we will update the path.

**Algorithm 1:** Disk Test

**Data:** input path $P$ as a set of vertices
**for** *every vertex* $v_i$ *in* $P \setminus (v_0, v_n)$ **do**
  $d_i \leftarrow \min_O D(v_i, O)$
**end**
**if** *TerminationConditionsAreMet()* **then**
  return done
$r \leftarrow \max(d_i)$
$v^* \leftarrow P[index(\max(d_i))]$
$s = (\mathbf{p}_1, \mathbf{p}_2) \leftarrow$ PointsOfCircleIntersection($r$, $v^*$)
return $s$

When there are fewer than two points of intersection between the circle and the path, then the circle must encompass one or both of the endpoints of the path. Therefore, in such cases, we choose those encompassed path endpoints as the endpoints of the shortcut. Additionally, if there are more than two intersections, then we choose the earliest and latest among those intersections, with respect to each path direction, as the shortcut endpoints.

The overall Disk Test procedure is summarized in Algorithm 1.

### 3.4.2 Corner Test

The Corner Test has similarities with the Disk Test; however, the key difference is that it only considers a subset of the obstacles. For every vertex, excluding the

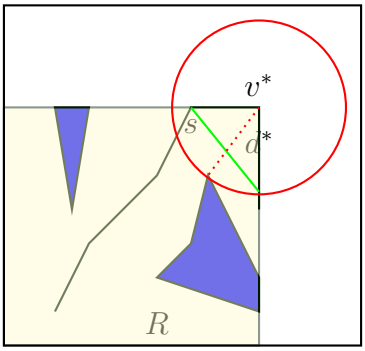


Figure 3.1: *An example of the Corner test: an environment including two obstacles, $v^*$ the vertex with the most free space (largest radius $d_i$ called $d^*$), the convex region $R$ in yellow, and shortcut $s$ in green.*

start and end vertices, a *corner* is defined as the triplet consisting of the previous vertex, the current vertex, and the next vertex. The Corner Test computes the same distance to the obstacles as with the Disk Test, except that now only obstacles that are inside of the *corner region* defined by the corner are considered, as defined below.

Given a corner $(\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2)$, we define the corner region as the region in-between the rays $(\mathbf{v}_1, \mathbf{v}_0)$ and $(\mathbf{v}_1, \mathbf{v}_2)$. While the Corner Test searches for obstacles at any distance from $\mathbf{v}^*$, it limits our choice of shortcut endpoints to be inside the sub-path delimited by rays $(\mathbf{v}_1, \mathbf{v}_0)$ and $(\mathbf{v}_1, \mathbf{v}_2)$.

When the number of intersections between the circle and the path is not exactly 2, we choose the shortcut endpoints in the same way as described for the Disk Test. However, when applying the Corner Test, we limit the shortcut endpoints to not exceed the previous and next vertices of $\mathbf{v}^*$. The pseudo-code for the procedure is given in Algorithm 2.

In Figure 3.1, the region $R$, shown here in yellow, is the only area where obstacles are considered. A shortcut $s$ is then built from the intersection of $C$ and path $P$. With the Corner Test the generated shortcut is not allowed to go outside the corner region. Figure 3.2 shows an example iteration of the procedure when applied to some initial path.

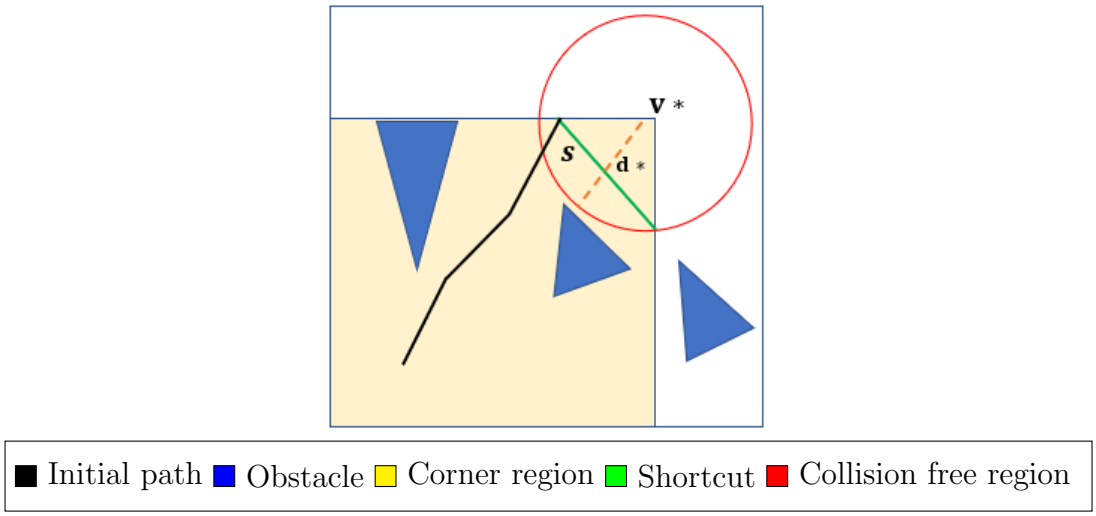


Figure 3.2: *An example of the Disk test: an environment with two obstacles, $v^*$, the vertex with the most free space (largest radius $d_i$ called $d^*$), circle in red, and shortcut $s$ in green.*

### 3.4.3 Termination Condition

We define a termination condition based on the quality of the solution, according to two criteria. When one of the two quality criteria are met for every vertex, the smoothing iterations stop and the algorithm terminates.

The first criterion determines, for a given vertex $v$, if $v$ is within the threshold distance to any obstacle in the environment. This means that $v$ is already at the limit distance to the obstacles and cannot be further optimized.

The second criterion is a measure of smoothness around $v$ that is simply based on the angle between the two path edges sharing $v$. If the angle is larger than a desired threshold, then this criterion is met for $v$.

When these criteria are met for a given vertex it means that the vertex is not suitable for optimization, and will not be chosen as $\mathbf{v}^*$. When every vertex satisfies at least one of these criteria, then the optimization terminates.

### 3.4.4 DSS Method

At each iteration of our proposed DSS method one of the two shortcut determination tests proposed in the previous subsections is employed. Figures 3.3 and 3.4

**Algorithm 2:** Corner Test

**Data:** input path $P$ as a set of vertices

**for** *every vertex* $v_i$ *in* $P \setminus (v_0, v_n)$ **do**

  $Y \leftarrow$ subset of $O$ that is in corner region

  $d_i \leftarrow \min_Y D(v_i, Y)$

**end**

**if** *TerminationConditionsAreMet()* **then**

  return done

$r \leftarrow \max(d_i)$

$v^* \leftarrow P[index(\max(d_i))]$

$s = (\mathbf{p}_1, \mathbf{p}_2) \leftarrow$ PointsOfCircleIntersection($r$, $v^*$)

return $s$

illustrate cases in which either the Corner or Disk Test can be most advantageous.

We first consider the Corner Test because we have found it to be, most of the time, the better operation to perform. Considering Figure 3.3, it is possible to see that when the obstacles in the corner region are far away, the shortcuts tend to be longer. In this case, the Corner Test makes a more useful shortcut than the Disk Test, since there might be obstacles not inside corner region which are closer to $\mathbf{v}^*$. On the other hand, if the distance from the obstacles to $\mathbf{v}^*$ is the same on both sides of the path, or the closest obstacle is in the corner region, then it is more advantageous to employ the Disk Test.

Therefore we first check if the Corner Test provides an effective shortcut, and if not, we then compute the result of the Disk Test and compare the two obtained shortcuts in order to select the best one. The pseudocode for DSS is given in Algorithm 3 (we note here that the termination conditions are handled in the Corner Test and Disk Test functions). Finally, note that the order in which the vertices are processed may affect the determination of $\mathbf{v}^*$.

Using a single test (Corner or Disk) does not always perform well in terms of length, sharpest angle, and average angle, which are our metrics of interest. Therefore, we define parameters $\delta$ and $k$ to specify how DSS should decide which

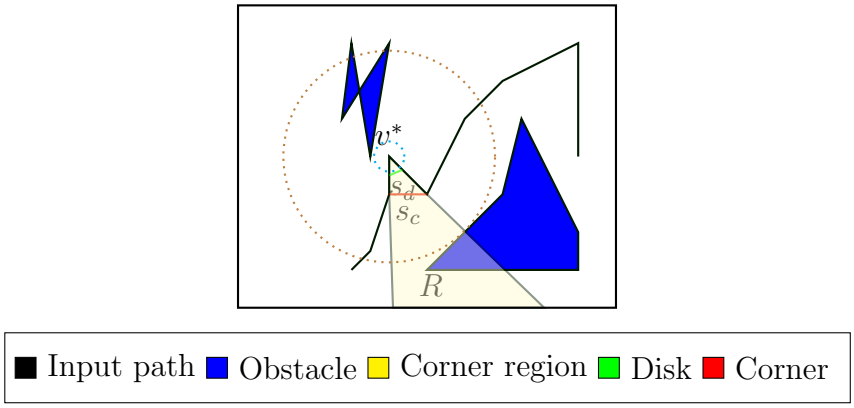


Figure 3.3: *Comparative example between shortcuts* $\mathbf{s_c}$ *(in red) and* $\mathbf{s_d}$ *(in green) obtained, respectively, with the Corner Test and the Disk Test. Here the Corner Test provides the longest shortcut because it only considers obstacles inside the corner region* $R$*.*

method is better at the current iteration. Parameter $k$ represents a factor applied to the radius of the circle used to derive the Corner Test's shortcut. When the length of the shortcut is much smaller than the radius of the circle, DSS chooses to also consider the shortcut provided by the Disk Test. Since no chord of a circle can be larger than the diameter, it does not make sense to choose $k > 2$. Parameter $\delta$ provides a similar discrimination, but according to the absolute distance of the shortcut from the Corner test, rather than its ratio to the radius. Regardless of the used parameter values, if the Disk Test is considered, then its shortcut is compared with the shortcut obtained from the Corner Test, and the shortcut with greater length is ultimately used.

### 3.4.5 Random shortcut Method

Random shortcut heuristic methods replace portions of a path with shorter segments in the configuration space, and check if they are collision free. The pseudocode showing the implementation we used for our evaluation, while staying close to the canonical form of the method in continuous environments is provided in algorithm 4.

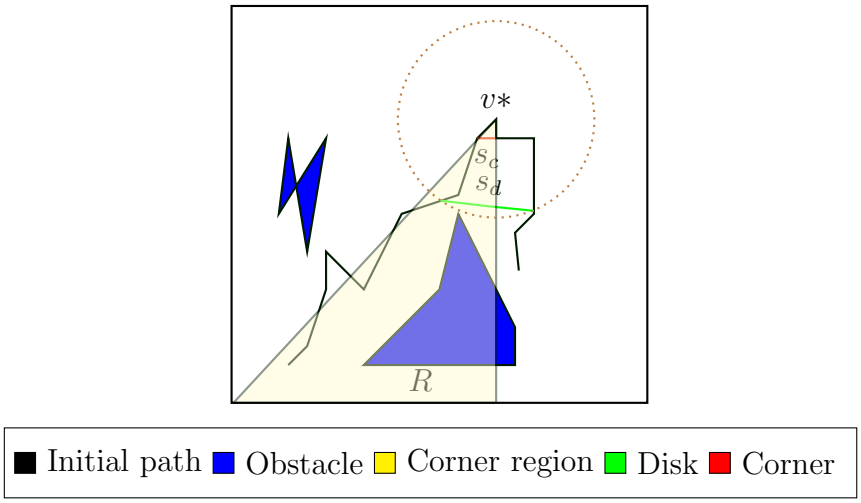


Figure 3.4: *Comparative example between shortcuts* $\mathbf{s_c}$ *(in red) and* $\mathbf{s_d}$ *(in green) obtained, respectively, with the Corner Test and the Disk Test. The corner region R is shown in yellow. Here the Disk Test provides the longest shortcut.*

**Algorithm 3:** DSS ($\delta$, $k$)

**Data:** $\delta$, $k$: Selection parameters, $O$: Obstacles, $P$: Current path

InitializeEnvironment(P, O)

$s_1$ = CornerTest()

**if** $s_1.length < \delta + s_1.r \cdot k$ **then**

  $s_2$ = DiskTest()

  **if** $s_2.length > s_1.length$ **then**

    UpdatePath($P$, $s_2$)

  **else**

    UpdatePath($P$, $s_1$)

  **end**

**else**

  UpdatePath($P$, $s_1$)

**end**

**Algorithm 4:** Random Shortcut

**Data:** initial path as set of vertices $P$, execution time $T$

**while** *current time* $< T$ **do**

  Choose vertices $v_i$, $v_j$ uniformly without replacement from $P \setminus t$

  Choose $r_i$ uniformly along the edge $(v_i, v_{i+1})$

  Choose $r_j$ uniformly along the edge $(v_j, v_{j+1})$

  **if** *collisionFree($r_i$, $r_j$)* **then**

    updatePath(P, $(r_i, r_j)$)

**end**

## 3.5 Evaluation and results

We validated our DSS method and compared it against a regular implementation of the Random Shortcuts method in five different environments which contained a variety of convex and non-convex obstacles of different sizes and placements. Both methods only address static obstacles.

Our implementation of the Random Shortcuts is based on: 1) sampling random pairs of points along the current path, 2) checking if the shortcut connecting a pair of points is collision-free and respecting the given minimum clearance, and 3) if that is the case, the respective path section is replaced with the sampled shortcut. This implementation reflects how the approach is mostly used, or cited, in previous work [39].

We performed 50 trials for each environment. For every environment, a trial consisted of: (1) randomly choosing start and goal locations, (2) optimizing a path result computed with an RRT implementation, and (3) optimizing the path using the methods being compared.

The three metrics we used to compare the algorithms, which are common in the literature [8, 82], are *average angle*, *sharpest angle* and *average length*. The first two metrics capture how smooth the final solution is, while the last metric captures the cost of traversal for the final solution. If the average angle is larger, the path is considered smoother. These metrics are useful to determine a path that is easier and faster to traverse for robots with typical dynamic constraints [86].

For comparison, we implemented the Random Shortcut method and ran it for the same amount of time as the DSS method.

For comparison, we implemented a Randomized shortcut method and ran it for the same amount of time as the DSS method, which has its own termination conditions. We add to our analysis by considering the theoretical running time of DSS in comparison to the Random Shortcut technique. Let $N$ be number of vertices in initial path, and let $L$ be the total number of edges in all obstacles. Since the random shortcut method must check if a given shortcut is collision free, each iteration of random shortcut takes $O(L)$ running time. Each iteration of our method must find the distance between each vertex and the nearest obstacle which

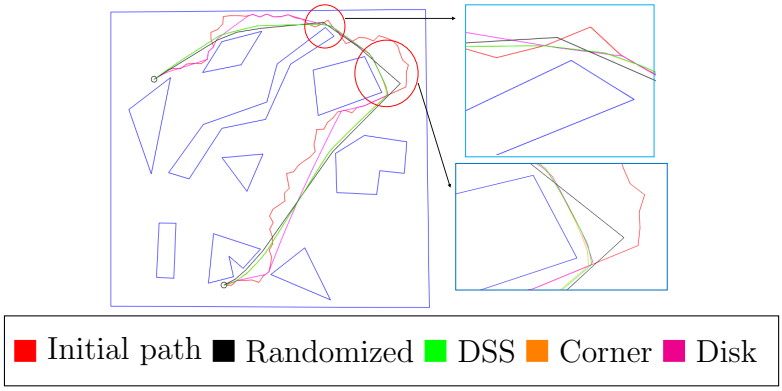


Figure 3.5: *Results produced by DSS, DSS with only the Corner Test, DSS with only the Disk Test and Random Shortcuts in an environment with obstacles of diverse shapes.*

we compute by iterating over every pair. Therefore, each iteration of our method takes $O(NL)$ running time. However, the probability of the shortcut proposed by the random method being collision free is strongly dependent on the initial path and the environment. Our method always chooses the vertex with the most free space for optimizing. Therefore, the expected number of iterations of the random method that would be needed to achieve the same stopping conditions as our method achieves is much greater than the number of iterations that our method requires. This analysis agrees our experimental results comparing the average angle, path length and sharpest angle achieved by the two methods when run for the same amount of time.

Figure 3.5 represents a visualization of DSS versus Random Shortcuts in one of our test environments. We circled and zoomed the regions of interest. Figure 3.5 shows that DSS produces a final path in green which is smoother and shorter than the final path obtained by regular Random shortcuts. Due to the sharpest angle in the final path produced by Random Shortcuts (method shown in black), we can say DSS is smoother.

Figure 3.6 shows a similar improvement as in figure 3.5 when using the DSS method.

Table 3.1 shows the performance for $\delta = 2.0$ and $k = 0.0$. In almost all scenarios, DSS performs better in terms of average angles and sharpest angles, as

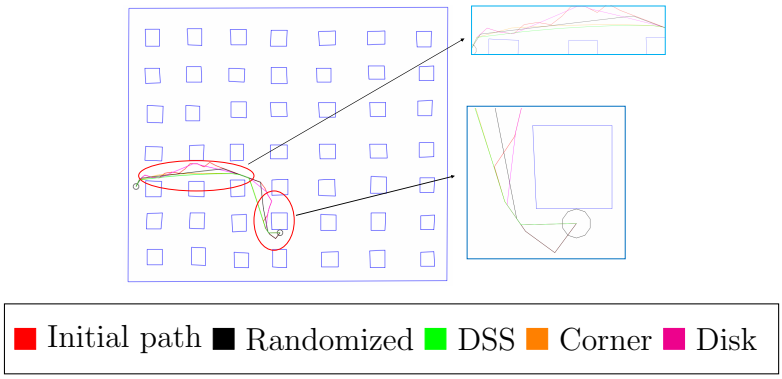


Figure 3.6: *Results produced by DSS, DSS with only the Corner Test, DSS with only the Disk Test and Random Shortcuts in an environment with regularly spaced obstacles.*

well as average length. DSS with only the Disk Test performs better than DSS with only Corner Test in the Simple environment, as expected according to the discussion proposed in Section 3.4.

While our current method proves to be more effective than the regular random selection of shortcuts, a number of additional combinations of the proposed deterministic tests and selection parameters can be explored which we however leave for future work.

## 3.6 Conclusion

We show that the proposed priority-based deterministic shortcut method, for the same amount of computation time, produces comparable and in many cases better results than the regular random selection of shortcuts in terms of path length and smoothness.

In general, the Corner Test is more advantageous since it can safely ignore some of the obstacles in the environment, however the Disk Test is more effective in particular cases. A promising future work is to include a characterization of the complexity of the environment in relation to the performance of either the Corner Test or the Disk Test.

More generally this work shows that simple geometric tests can improve the

| Metric | Method | Environment | | | | |
|---|---|---|---|---|---|---|
| | | Mixed | Interlocked | Regular | Simple | Office |
| Avg Angle | Random | 152.34 | 143.99 | 149.46 | 141.93 | 143.31 |
| | Corner | 174.18 | 175.39 | 173.52 | 166.63 | 173.15 |
| | Disk | 169.29 | 169.41 | 173.52 | **172.74** | **173.93** |
| | DSS | **174.68** | **175.41** | **173.67** | 166.54 | 172.95 |
| Sharpest Angle | Random | 88.60 | 80.10 | 81.25 | 83.9 | 83.39 |
| | Corner | 144.11 | 147.86 | 138.63 | 131.20 | **144.95** |
| | Disk | 127.74 | 132.28 | 123.87 | **137.49** | 143.14 |
| | DSS | **145.23** | **147.93** | **139.99** | 130.15 | 144.88 |
| Avg Length | Random | 16.34 | 17.47 | 15.10 | **18.05** | **15.26** |
| | Corner | 16.43 | **17.44** | 15.05 | 19.35 | 15.59 |
| | Disk | 16.97 | 18.16 | 15.47 | 18.34 | 15.57 |
| | DSS | **16.27** | 17.52 | **14.98** | 19.42 | 15.57 |

Table 3.1: Table showing the different methods. Here DSS is called as DSS($\delta = 2.0, k = 0$). DSS performs better than the Randomized shortcut method in several environments.

performance of shortcut-based path smoothing techniques, motivating further developments in this area. We intend to further improve our path optimization method by employing Bézier curves as path segments in order to produce $C^2$ continuity and the opportunity to address curvature constraints to the resulting optimized paths.

# Chapter 4

# Optimizing Curvature and Clearance of Piecewise Bézier Paths

## 4.1 Introduction

Path planning is an important procedure in robotics and computer graphics, among other areas. While this focuses on applications related to autonomous vehicles, path planning can be applied to generic configuration spaces under different types of constraints, and is also an important topic in robotic manipulation [87].

Many researchers have investigated path smoothing and optimization methods, which generally fall into one of two main categories: iterative geometric improvement, or optimization using linear, convex, or non-convex models. Our proposed work is related to the second category as we employ a convex optimization approach for our proposed model.

Our method is applied to paths represented as piecewise Bézier Curves [90]. Bézier curves and other forms of Spline curves are very popular for path representation [59,63,65,90], since they are parameterized and, in comparison to other polynomial interpolation approaches, Runge's phenomenon can be avoided for higher degrees. B-spline curves also represent a popular approach for Spline-based path

or trajectory representation [28, 77, 78]. We rely on Bézier curves because of their simpler formulation and suitability for integration in our optimization framework.

In this chapter we propose a path optimization method based on convex optimization, where we focus on addressing clearance constraints while optimizing the length and curvature of a path represented as a piecewise quadratic Bézier curve. Addressing curvature is important for ensuring that the resulting path avoids sharp turns. For instance, such curves are difficult to be followed by mobile robots.

We apply our method to optimize low-quality polygonal paths generated by a Rapidly-Exploring Random Tree (RRT) planner [61]. The sampling-based nature of this planner generates paths that are not usable without smoothing procedures [1, 101].

Our approach can be applied to optimize such paths. More in general our method addresses applications that require smooth paths considering curvature and clearance constraints.

We propose a piecewise Quadratic Bézier Convex Optimization method (QBCO) which allows users to customize terms in the objective function for controlling clearance, length, and curvature. This process includes a method for transforming a piecewise linear path, which is given as input, into a piecewise quadratic Bézier path. We then present a set of optimization constraints which ensure $C^1$ continuity and how our objective function can be written in order to address length, curvature and clearance.

We provide several benchmarks to show the results of our method, and also present comparisons against a shortcut-based smoothing method. Our method is fast and gives users the flexibility to customize the properties of the obtained paths.

## 4.2 Related Work

Path optimization is often applied to smooth paths generated by motion planners. In particular, sampling-based planning techniques [62] have sparked a demand for effective path smoothing algorithms.

These methods have become very popular primarily because of their ability to solve problems in high-dimensional spaces [1, 101]. Sampling-based algorithms however produce

paths that often feature numerous twists and turns, making it necessary to incorporate a post-processing smoothing step. In order to evaluate our results we apply our path optimization method to paths generated by a sampling-based RRT planner [61].

Direct trajectory optimization methods [5, 26, 100] can be applied to optimize high-dimensional trajectories, with consideration given to addressing the robot kinematics and dynamics. However, when such planning problems are transcribed as non-convex programs of local optimization scope, these methods can be unsuccessful in discovering a collision-free trajectory, particularly in cluttered configuration spaces.

Optimizers based on geometric path representations are also popular. Choi et al [16, 17, 19] have presented path planning algorithms that utilize Bézier curves for autonomous vehicles with waypoint and corridor constraints. Similar to our work, their algorithms generate paths for vehicles from a series of Bézier curve segments. These methods use a constrained optimization technique that aims to minimize the curvature and length of the path, while maintaining $C^1$ (and in the most recent paper $C^2$) continuity. They ensure their path is collision-free by dividing the space into a series of convex regions within a given corridor of the environment. However, this approach requires a corridor to be computed and does not take into account customizable terms including both curvature an clearance in order to address arbitrary distance from obstacles. In contrast, our approach automatically extracts corridor information with a set of free disks centered on the path, which constitute our collision-free region for optimization. Our corridor definition adapts to the free space as the optimization is performed. Another limitation of [17] is that the control points of the Bézier curve are forced to be in the free space. Since not all of the control points of a Bézier curve fall on the curve itself, we provide the path more freedom by allowing control points to fall outside the free space as long as the path is collision-free. Finally, our formulation allows

the user to customize the weights influencing the length, curvature, and clearance terms of our objective function.

Also related to our work is the geometric approach of Cimurs et al [21], where a path smoothing method also using Bézier curves is proposed. Their overall method consists of four modules: node generation by the path planner, shortest path selection, removal of unnecessary nodes using a shortcut method, and node alignment. Although their method can produce smooth short paths, does not use optimization software and does not consider curvature as part of the smoothing procedure.

Additionally, their approach uses the convex hull of the control points to check for collisions, which is overly conservative. Our method utilizes disks centered on the path in order to define a collision-free optimization region without sacrificing performance.

Following a geometric approach, Geraerts et al [34] have proposed several techniques to improve the quality of paths by shortening their length and maximizing their clearance. They propose an iterative technique to increase the clearance of a path in order to improve its computation time.

However, the proposed method does not address both clearance and curvature as part of the optimization procedure. In contrast our approach formulates the problem in a generic optimization scheme that addresses all these properties in a unified way.

There has also been work by Neto et al [76], which attempts directly generate a path composed of Piecewise Bézier curves using RRT, rather than converting from line segments to Bézier curves. This work insures curvature continuity using degree 7 polynomials, but does not offer any path optimization.

Kielas et al [57] present a technique that utilizes degree elevation of Bernstein polynomials to produce optimal trajectories. This method avoids collisions through finding the convex hull of a polynomial curve. By adding more control points to the curve, the convex hull can be brought closer to the curve without changing its shape. Although we do not focus on improved collision detection, such a collision detection technique could be integrated in our work.

| Work | Curvature | Clearance | Input | Bézier Degree | Collision Check | Termination |
|---|---|---|---|---|---|---|
| [17] | Min | - | Corridor | Any | Convex Hull | One Round |
| [21] | - | Min | Voronoi | Cubic | Convex Hull | One Round |
| [15] | - | - | Corridor | Cubic | Convex Hull | One Round |
| QBCO | Min | Min | RRT | Quadratic | Sequence of disks | Convergence |

Table 4.1: Summary of properties observed in related work and our proposed method. Our method is the only one minimizing curvature and length while controlling clearance. Our method also has a more accurate way to model collision checks.

Table 4.1 presents a summary comparing the most relevant methods discussed above.

## 4.3 Mathematical model and Definitions

Bézier Curves were invented in 1962 by the French engineer Pierre Bézier for designing automobile bodies. Today, Bézier Curves have significant applications in computer graphics and animation [25, 72]. According to [15], a Bézier Curve of degree $n$ can be represented as

$$P(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{P_i}$$

Where $\mathbf{P_i}$ are control points such that $P(0) = \mathbf{P_0}$ and $P(1) = \mathbf{P_n}$, $B_i^n(t)$ is a Bernstein polynomial given by

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i, \quad i \in \{1, ..., n\}$$

Some of the useful properties of Bézier Curves for path planning are listed below:

- They always passes through control points $\mathbf{P_0}$ and $\mathbf{P_n}$.
- They are always tangent to the lines connecting $\mathbf{P_0} \rightarrow \mathbf{P_1}$ and $\mathbf{P_n} \rightarrow P_{n-1}$ at $\mathbf{P_0}$ and $\mathbf{P_n}$ respectively.
- They always lie within the convex hull consisting of their control points.

A quadratic Bézier curve $P$ is constructed by three control points $\mathbf{P_0}, \mathbf{P_1}$, and $\mathbf{P_2}$:

$$P(\lambda) = \mathbf{P_0}(1-\lambda)^2 + 2\lambda(1-\lambda)\mathbf{P_1} + \lambda^2\mathbf{P_2} \tag{4.1}$$

Where $\lambda \in [0, 1)$ represents the proportion of the distance along the curve between $P_0$ and $P_2$.

**Derivative, Continuity and Curvature of Bézier Curve** The derivatives of a Bézier curve can be determined geometrically from its control points [15, 88]. The first derivative of a Bézier curve

$$P(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{P_i}$$

is evaluated as

$$P'(t) = \sum_{i=0}^{n-1} n(\mathbf{P_{i+1}} - \mathbf{P_i})B_i^{n-1}(t) \tag{4.2}$$

Where $n(\mathbf{P_{i+1}} - \mathbf{P_i})$, are control points of $P'(t)$

In order to obtain the higher order derivative of a Bézier curve, the relationship of equation 4.2, can be used iteratively.

Two Bézier curve $P(t)$ and $Q(t)$ are said to be $C^k$ continuous at $t_0$ if

$$P(t_0) = Q(t_0), \quad P'(t_0) = Q'(t_0), ..., P^k(t0) = Q^k(t0) \tag{4.3}$$

Thus, $C^0$ continuity means simply that the two adjacent curves share a common endpoint. $C^1$ continuity means that the two curves not only share the same endpoint, but also that they have the same tangent vector at their shared endpoint, in magnitude as well as in direction. $C^2$ continuity means that two curves have $C^1$ continuity and, in addition, that they have the same second order parametric derivatives at their shared endpoint, both in magnitude and in direction. Figure 4.1 shows a demonstration of simple, quadratic and cubic Bézier curves.

**Lemma 1** If $P(t)$ and $Q(t)$ are at least $C^2$ continuous at $t_1$, for the path constructed by two Bézier curve segments $P(t)$ and $Q(t)$, then the path has continuous curvature for every point on it.

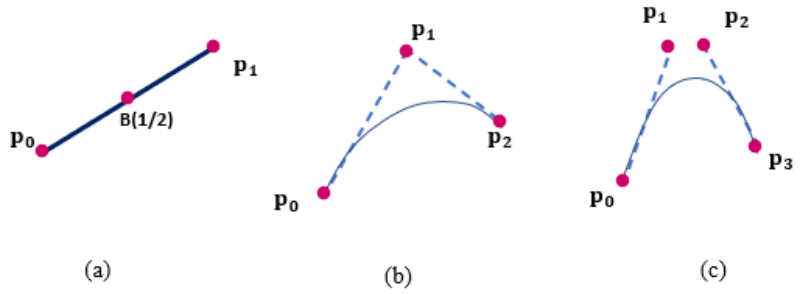


Figure 4.1: *(a) shows a linear Bezier curve. (b) shows a quadratic Bezier curve and (c) shows a cubic Bezier curve.*

## 4.4 Method

### 4.4.1 Problem Statement

We are given a path as an ordered set of tuples $P = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_N\}$, $\mathbf{v}_i \in \mathbb{R}^2$, and a set of polygonal obstacles $O = \{O_1, O_2, ..., O_M\}$. Our goal is to optimize the path between the initial and final points of $P$. We call $P$ a piecewise linear path. Our optimization criteria are curvature, clearance, and path length. More specifically, we enforce $C^1$ continuity while minimizing a combination of the maximum curvature along the path, the distance between the path and obstacles, plus the length of the path, subject to user-specified coefficients. In the scope of this work, we address the particular case of 2-dimensional polygonal paths, where each vertex is a tuple $\mathbf{v}_i = (x_i, y_i)$, and each obstacle is an ordered set of tuples $O_i = \{\mathbf{o}_{i,1}, \mathbf{o}_{i,2}, ...\}$.

Our proposed method first transforms the input path to an ordered set of Quadratic Bézier curves $P' = \{B_1, B_2, ...\}$, where each quadratic Bézier curve is defined by three control points $B_i = \{\mathbf{p}_{i,0}, \mathbf{p}_{i,1}, \mathbf{p}_{i,2}\}$. Our method therefore computes and improves the positions of the control points which define a continuous collision-free path. In this section, we also use the notation $B(t)$ to refer to a given quadratic Bézier curve evaluated at $t$.

### 4.4.2 Generating the Initial Piecewise Bézier Path

**Description of methods and evaluation criteria**

**Approach 1**: **Quadratic Bézier approach** This chapter proposes an efficient, quadratic Bézier curve-based approach providing $C^1$ curvature continuity for path planning in a static environment. The Bézier curve-based algorithm will be obtained by the following steps:

1. At every iteration we optimize the path at the vertex with the most free space. First, we start by calculating the minimum Euclidean distance to every obstacle in the environment, for every non-terminal vertex:

$$d_i = \min_{O_j \in O} D(\mathbf{v_i}, O_j), \quad \forall i = 2, 3, ..., N-1$$

   Where $O$ is the set of obstacles, $O_j$ is an obstacle in the environment and $D$ is a function that returns the minimum distance from an obstacle to a vertex in the path. This distance serves as a proxy for the free space available at a vertex. We determine the vertex with the most free space, which we call $\mathbf{v}^*$, and its corresponding distance $d^*$. In other words, $\mathbf{v}^* = \mathbf{v_i}$, $d^* = d_i$ such that $\max_{j \in [N]} d_j = d_i$. The distance $d^*$ is used as the radius of a circle centered at $\mathbf{v}^*$, which we call $C$. The points of intersection between $C$ and the path form the endpoints of the shortcut with which we will update the path.

2. We use the same intuition in the Corner test, but now we only consider a subset of the obstacles inside of the corner. Given a corner $(\mathbf{v_0}, \mathbf{v_1}, \mathbf{v_2})$, we define the convex region $R$ as the area in-between the rays $(\mathbf{v_1}, \mathbf{v_0})$ and $(\mathbf{v_1}, \mathbf{v_2})$. Then we apply Corner test in order to get the initial shortcut.

3. In order to apply the Bézier curve, we consider all possible cases that happen for the obtained shortcut from Corner test. More clearly, the endpoints of the shortcuts obtained by the Corner test follow two cases:

   (a) Case 1: The shortcut's endpoints are inside convex region. We label the shortcut endpoints $\mathbf{P_0}$ and $\mathbf{P_2}$. We create a quadratic Bézier curve with control points $(\mathbf{P_0}, \mathbf{v}^*, \mathbf{P_2})$.

(b) Case 2: One of the shortcut's endpoints are outside convex region. We label the endpoint inside the convex region $\mathbf{P_0}$. We then label the midpoint between $\mathbf{v}^*$ and the vertex following $\mathbf{v}^*$ as $\mathbf{P_2}$. We create a quadratic Bézier curve with control points $(\mathbf{P_0}, \mathbf{v}^*, \mathbf{P_2})$.

(c) Case 3: Both shortcut endpoints are outside convex region. Similar to case 2, we label the midpoints between $\mathbf{v}^*$ and its previous and next vertices $\mathbf{P_0}$ and $\mathbf{P_2}$, respectively. We create a quadratic Bézier curve with control points $(\mathbf{P_0}, \mathbf{v}^*, \mathbf{P_2})$.

Figure 4.2 shows the resulting shortcut from each case of the quadratic Bézier curve selection process explained above.

We choose the first and last control points of the quadratic Bézier curve, which we also call the end points of the shortcut, carefully in order to avoid scenarios in future iterations where it is not possible to choose a quadratic Bézier curve which maintains $C^1$ continuity.

In all cases, none of the endpoints of the Bézier curve shortcut are allowed to go beyond the previous and next vertices of $\mathbf{v}^*$ because we are not checking all the obstacles in the environment. Bézier curves are fast to compute, therefore, by using these curves, we will not lose too much performance compared to DSS method.

Following the optimization process, our findings revealed that the geometric rules we had created for converting linear shapes into Bézier curves had a minimal influence on the outcomes of my testing. Consequently, we decided to use the most simple version of linear to Bézier for all our work going forwards.

**Approach 2**: To create a smooth (e.g. $C^1$ continuous) path from a piecewise linear path, we devise a simple curve fitting technique which involves creating one quadratic Bézier curve for each vertex of the input polygonal path. For each vertex $\mathbf{v}_i \in P, i \neq 1, N$, we define the control points for curve $B_i = \{\mathbf{p}_{i,0}, \mathbf{p}_{i,0}, \mathbf{p}_{i,0}\}$ in $P'$ as:

- $\mathbf{p}_{i,0} = 0.5(\mathbf{v}_{i-1} + \mathbf{v}_i)$,
- $\mathbf{p}_{i,1} = \mathbf{v}_i$,

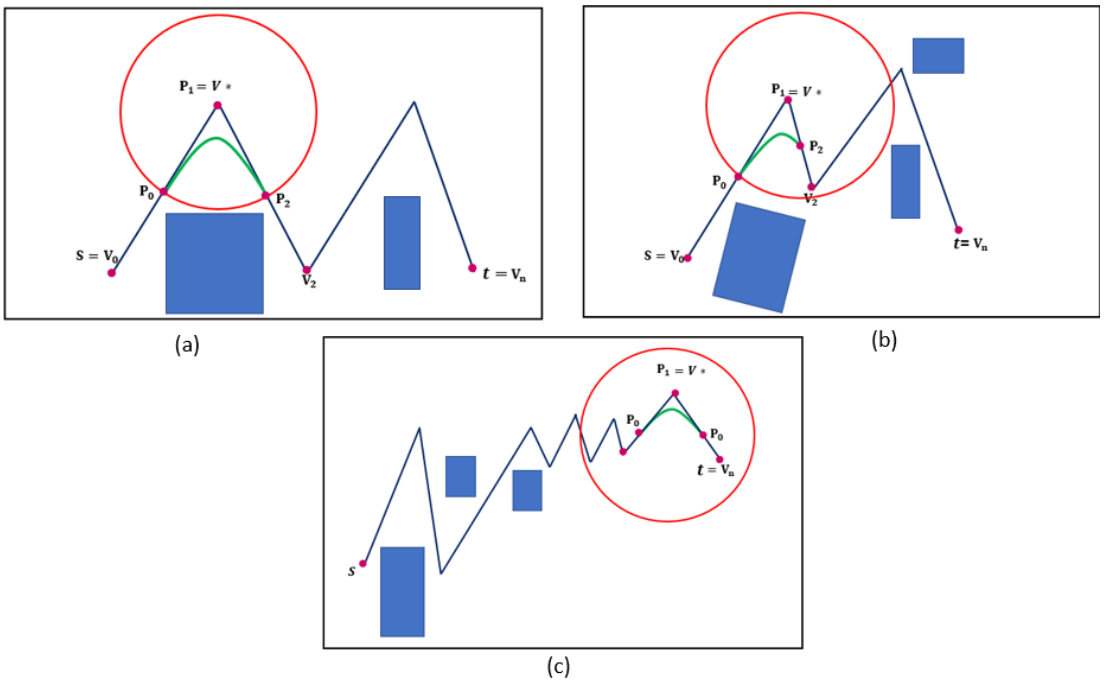


Figure 4.2: *(a) The intersections between the circle and the path are selected as the end points of the shortcut and are inside the convex region. (b)When one or more of the intersections between the circle and the path fall outside the convex region, we choose the midpoint between* $\mathbf{v}^*$ *and the next vertex inside the convex region as the end point of the shortcut on that side.(c) when there are fewer than two intersections between the circle and the path, we can treat this scenario similarly as when the intersection are outside the convex region.*

- $\mathbf{p}_{i,2} = 0.5(\mathbf{v}_i + \mathbf{v}_{i+1})$.

Thus, $\mathbf{p}_{i,0}$ is the midpoint between $\mathbf{v}_i$ and the previous vertex, while $\mathbf{p}_{i,0}$ is the midpoint between $\mathbf{v}_i$ and the next vertex. For the first vertex, $\mathbf{p}_{0,0} = \mathbf{v}_0$, and for the last vertex, $\mathbf{p}_{N-1,2} = \mathbf{v}_N$.

This process will not introduce any collisions with the environment if the initial path $P$ already has enough clearance from obstacles, because the distance between $P$ and $P'$ after this operation is very small.

However, in the case when the initial clearance is not sufficient, we may check each Bézier curve for collisions. If a collision is found for a given curve, the first and last control points of the curve are moved to the mid-points between their original positions and vertex $\mathbf{v}_i$, effectively making the curve to be closer to the

input polygonal path at vertex $\mathbf{v}_i$. When this binary subdivision is performed, two new Bézier curves forming straight segments are added to connect the created gap at the beginning and end of the modified curve. This process can be repeated recursively in order to ensure that no collisions are introduced when converting the input path to the piecewise Bézier representation.

### 4.4.3 Optimization Variables

Let $\{\mathbf{m}_{i,0}, \mathbf{m}_{i,1}, \mathbf{m}_{i,2}\} = B_{m,i} \in M, \forall i$, be variables for the optimization model $M$, where $\mathbf{m}_{i,j} = (x_{m,i,j}, y_{m,i,j})$. Each $\mathbf{m}_{i,j}$ corresponds to $\mathbf{p}_{i,j} \in P'$. We can evaluate the model variables just like we do with the control points:

$$B_{m,i}(t) = (1-t)^2\mathbf{m}_{i,0} + 2t(1-t)\mathbf{m}_1 + t^2\mathbf{m}_{i,2}. \tag{4.4}$$

### 4.4.4 Multi-Objective Optimization Function

Since there is more than one independent quantity to optimize, our problem requires multi-objective optimization. We assign a weight (or coefficient) to each objective term to indicate its relative importance in the model. Thus we write our objective function for max curvature $K_{\max}$, clearance $\Delta$, and length $L$ as:

$$\text{Minimize} \qquad c_K \sum K_{\max} + c_\Delta \Delta + c_L L. \tag{4.5}$$

The maximum curvature along a single quadratic Bézier curve can be calculated with:

$$K_{Max} = \frac{\|\mathbf{p}_2 - 2\mathbf{p}_1 + \mathbf{p}_0\|}{2|(\mathbf{p}_1 - \mathbf{p}_0), (\mathbf{p}_2 - \mathbf{p}_1)|}. \tag{4.6}$$

The mathematical procedure for obtaining this formula is presented in Appendix 7. This formula however is not suitable for optimization packages, since it is non-linear, non-convex, and discontinuous. Therefore, we approximate the max curvature using a first-degree Taylor Expansion, therefore linearizing the equation. The linearization involves taking derivatives of $K_{\max}$ with respect to each of the

optimization variables, evaluated at the corresponding control points of $P'$, which we take as our *base*. We apply the process of linearizing a function $y = f(x)$ with:

$$\hat{y} = (x - x_{base}) \left.\frac{df}{dx}\right|_{x=x_{base}} + f(x_{base}). \tag{4.7}$$

Where $\hat{y}$ is the linear approximation for $y$. Applying this process to $K_{\max}$ results in:

$$\begin{aligned} \hat{K}_{\max} = (x_0 - x_{0,base}) \left.\frac{dK_{\max}}{dx_0}\right|_{x=x_{0,base}} \\ + (y_0 - y_{0,base}) \left.\frac{dK_{\max}}{dy_0}\right|_{y=y_{0,base}} \\ + ... \\ + K_{\max}(x_{0,base}, y_{0,base}, ..., y_{2,base}). \end{aligned} \tag{4.8}$$

Where $\hat{K}_{\max}$ is the linear approximation for curvature. Expressions for each derivative of $K_{\max}$ can be found in the Appendix. This linearized version of max curvature is implemented in the objective function, substituting $\mathbf{p}_i$ with the corresponding optimization variable $\mathbf{m}_i$. We sum the max curvature of every curve along the path in order to minimize the maximum curvature throughout the path, rather than myopically focusing on only one curve at a time.

We represent the length of the entire piecewise Bézier curve as $L = \sum L_i$, where $L_i$ is the length of curve $B_i$. A quadratic Bézier curve $B_i(t)$ is defined on the range $t \in (0, 1)$. Therefore, we can express the arc length of this curve with:

$$L_i = \int_0^1 \left\| \frac{dB_i}{dt} \right\| dt. \tag{4.9}$$

A method for calculating the exact arc length for a quadratic Bézier curve is outlined in the Appendix. Alternatively, the arc length can be roughly approximated as the sum of the distances between its control points:

$$\hat{L}_i = \|\mathbf{p}_{i,0} - \mathbf{p}_{i,1}\| + \|\mathbf{p}_{i,2} - \mathbf{p}_1\|. \tag{4.10}$$

Where $\hat{L}_i$ is our approximation for $L_i$. This approximated version of arc length is sufficient for the purpose of minimizing the total path length, and has been

implemented in our objective function, substituting $\mathbf{p}_{i,j}$ with the corresponding optimization variable $\mathbf{m}_{i,j}$.

The exact equation for determining the arc length of a quadratic Bézier curve is however used for reporting the total path length obtained by our optimization procedure.

The exact arc length is derived as:

$$I(x = 1) - I(x = 0), \tag{4.11}$$

where $I$ is represented by the following expression:

$$I = w_1 \left( w_2 \cdot w_3 + \log |w_3 + w_2| \right), \tag{4.12}$$

with fractions $w_1$, $w_2$, and $w_3$ specified as:

$$w_1 = \frac{n_2}{8a^{\frac{3}{2}}}, w_2 = \frac{n_1}{\sqrt{n_2}}, \text{ and } w_3 = \sqrt{\frac{n_1^2}{n_2^2} + 1}, \tag{4.13}$$

where:

$$\begin{aligned}
n_1 &= 2ax - b, \\
n_2 &= 4ac - b^2, \\
a &= 4x_2^2 + 16x_1^2 + 4x_0^2 - 16x_2x_1 + 8x_2x_0 - 16x_1x_0, \\
&\quad + 4y_2^2 + 16y_1^2 + 4y_0^2 - 16y_2y_1 + 8y_2y_0 - 16y_1y_0, \\
b &= -16x_1^2 - 8x_0^2 + 8x_2x_1 - 8x_2x_0 + 24x_1x_0, \\
&\quad - 16y_1^2 - 8y_0^2 + 8y_2y_1 - 8y_2y_0 + 24y_1y_0 \\
c &= -8x_1x_0 + 4x_0^2 + 4x_1^2 - 8y_1y_0 + 4y_0^2 + 4y_1^2.
\end{aligned} \tag{4.14}$$

*Clearance* in our convex optimization problem refers to the minimum distance between the path generated by the optimization algorithm and any obstacles in the environment. To ensure that the system is safe, it is important to consider clearance in the optimization problem.

Clearance is expressed in the objective function as a pseudo indicator function where control points with adequate clearance have low cost, and control points outside of the collision-free region have very high cost.

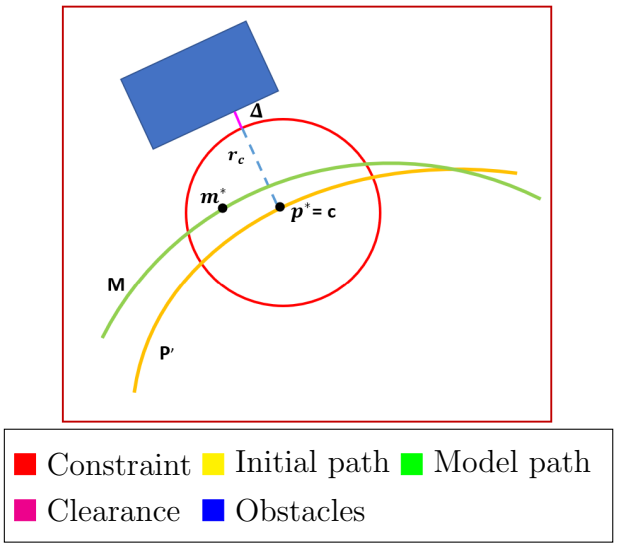


Figure 4.3: *Visualization of the Distance constraint*

### 4.4.5 Optimization Constraints

We optimize the model subject to the following constraints:

$$B_i(t=1) = B_{i+1}(t=0) \Rightarrow \mathbf{m}_{i,2} = \mathbf{m}_{i+1,0}, \tag{4.15}$$

$$B_i'(t=1) = B_{i+1}'(t=0)$$

$$\Rightarrow \mathbf{m}_{i,2} - \mathbf{m}_{i,1} - \mathbf{m}_{i+1,1} + 2\mathbf{m}_{i+1,0} = 0, \tag{4.16}$$

$$\|\mathbf{m}_{0,0} - \mathbf{v}_1\| < \epsilon, \tag{4.17}$$

$$\|\mathbf{m}_{N-1,2} - \mathbf{v}_N\| < \epsilon, \tag{4.18}$$

$$\|\mathbf{m}^* - \mathbf{c}\| < r_c. \tag{4.19}$$

$C^0$ **Continuity**: this constraint is expressed in equation 4.15. $C^0$ continuity means two adjacent (ordered one after the other) curves share an endpoint.

$C^1$ **Continuity**: equation 4.16 ensures $C^1$ continuity. This constraint means the tangent vectors of two adjacent curves must be equal at their shared endpoint. The resulting path therefore has smooth transitions between each consecutive pair of Bézier curves, with a continuous first derivative.

**Endpoints Constraint**: constraints 4.17 and 4.18 are related to the start and end points of the path. In our problem statement, we stated that the optimized

path $P'$ should maintain the same start and endpoint as the initial linear path $P$. If we enforce this exactly by constraining $\mathbf{m}_{0,0} = \mathbf{p}_{0,0}$ we would make the optimization model infeasible; therefore, we accept a small difference, $\epsilon$, in the position of the start and end of the path.

**Distance Constraint**: the distance constraint in equation 4.19 prevents collision between the path and obstacles, and is also used for the clearance objective. We identify a collision-free region around the path $P'$ by a collection of collision-free disks. The model $M$ is constrained to move the path only within the collision-free region. The center points of the disks are placed along the path, and the radii of the disks are set to be the distance between each path center and its closest obstacle.

To define these disks, we choose a set equally-spaced points along the path. For each chosen point, $\mathbf{p}^*$, we find the closest point $\mathbf{o}_{min}$, among the obstacles to $\mathbf{p}^*$:

$$\mathbf{o}_{min} = \underset{\mathbf{o} \in O_j, O_j \in O}{\arg\min} \|\mathbf{o} - \mathbf{p}^*\|,$$

with corresponding distance being:

$$d_{min} = \|\mathbf{o}_{min} - \mathbf{p}^*\|$$

We thus take the center of the collision-free disk corresponding to the chosen point to be equal to point $\mathbf{c} = \mathbf{p}^*$, with radius $r_c = d_{min} - \delta$, where $\delta$ is the desired clearance between the path and obstacles. The optimization model expression corresponding to $\mathbf{p}^*$ is $\mathbf{m}^*$. Precisely:

$$\begin{aligned} \mathbf{p}^* = B_i(t^*) &= (t^*)^2\mathbf{p}_{i,2} + 2t^*(1-t^*)\mathbf{p}_{i,1} + (1-t^*)^2\mathbf{p}_{i,0} \\ \Rightarrow \mathbf{m}^* &= (t^*)^2\mathbf{m}_{i,2} + 2t^*(1-t^*)\mathbf{m}_{i,1} + (1-t^*)^2\mathbf{m}_{i,0}. \end{aligned}$$

This disk is guaranteed to be collision-free, since the distance between $\mathbf{p}^*$ and $\mathbf{m}^*$ is constrained to be less than the distance between $\mathbf{p}^*$ and the nearest obstacle to $\mathbf{p}^*$. This constraint is illustrated in Figure 4.3. Since our environments and paths are 2-dimensional, we can further express constraint 4.19 as:

There exist several methods for calculating the distance between Bézier curves, such as those based on culling [11]. Using such methods may improve the running time of our proposed algorithm, but we leave this for future work.

$$(x_{m^*} - x_c)^2 + (y_{m^*} - y_c)^2 < r_c^2. \tag{4.20}$$

### 4.4.6 Convex Optimization Problem

To summarize, we have outlined the following convex optimization problem $M$:

$$\text{Minimize} \qquad \sum_{i=0}^{n-1} c_K \hat{K}_{\max_i} + c_\Delta \Delta_i + c_L \hat{L}_i \tag{4.21}$$

Subject to:

$$\mathbf{m}_{i,2} = \mathbf{m}_{i+1,0},$$
$$\mathbf{m}_{i,2} - \mathbf{m}_{i,1} - \mathbf{m}_{i+1,1} + 2\mathbf{m}_{i+1,0} = 0,$$
$$\|\mathbf{m}_{0,0} - \mathbf{v}_1\| < \epsilon,$$
$$\|\mathbf{m}_{N-1,2} - \mathbf{v}_N\| < \epsilon,$$
$$\|\mathbf{m}^* - \mathbf{c}\| < r_c.$$

Since our objective function is an approximation of the true max curvature and length, we chose to limit the distance between $\mathbf{m}_{i,j}$ and $\mathbf{p}_{i,j}$, $\forall i, j$, in addition to the above constraints. We point out that $\hat{K}_{\max}$ is the linear approximation for curvature and $\hat{L}_i$ is the quadratic approximation for length. This ensures that our approximations have small enough error to the true max curvature and length. To account for this limitation, we run the optimization program a few times, until convergence is reached. After each iteration, $P'$ is updated with the values calculated through convex optimization in $M$. The path is considered converged if the distance between the $P'$ and $M'$ is sufficiently small after an iteration. This process of running optimization multiple times also allows us to update the distance constraint before each iteration in order to precisely capture the free space available as the path changes shape. Algorithm 5 illustrates the pseudocode for this process.

**Algorithm 5:** Piecewise Quadratic Bézier Curve Convex Optimization (QBCO)

**Data:** Initial linear path $P$, Obstacles $O$

$P' \leftarrow$ QuadraticBezierMethod($P$)

$M \leftarrow$ CreateConvexModel()

**while** $P'$ *not converged* **do**

- AddVarsFromBezier($M$, $P'$)
- AddObjLinearCurvature($M$, $P'$)
- AddObjApproxLength($M$, $P'$)
- AddEndConstraints($M$, $P'$)
- AddC1Constraints($M$, $P'$)
- AddDistConstraints($M$, $P'$, $O$)
- Optimize($M$)
- $P' \leftarrow$ ExtractPath($M$)

**end**

## 4.5 Results and Evaluation

We have applied our method to optimize different types of input paths, in several environments and using varied sets of parameters.

### 4.5.1 Input Paths

To generate the initial piecewise linear path $P$ for optimization, we have used the popular RRT algorithm on different types of 2D environments. We ran several experiments using the path created from this method.

We used the shortcut-based path smoothing method demonstrated in [83] for comparison. This method, DSS, identifies the vertex with the most potential for path improvement at each iteration based on obstacle distance and corner properties in order to determinitstically create effective shortcuts to gradually improve the path. Termination occurs when all vertices are close to obstacles or the angles at vertices approach 180 degrees. DSS can also be used as a first pass to be applied

| Method | Time | Length | $K_{max}$ | $K_{ave}$ | Min Clear. | Ave Clear. |
|---|---|---|---|---|---|---|
| RRT | <0.01 | 21.48 | - | - | 0.50 | 1.53 |
| QBCO (Length) | 5.36 | **18.78** | 2.96 | 2.24 | 0.25 | 1.57 |
| QBCO (Curvature) | 0.59 | 34.83 | **2.65** | **1.12** | 0.38 | 1.43 |
| QBCO (Length + Curvature) | 0.75 | 20.20 | 3.70 | 2.32 | **0.87** | **1.82** |
| DSS | 0.11 | 12.22 | - | - | 0.04 | 0.76 |
| QBCO (Length) | 5.02 | **12.21** | 1.98 | 1.25 | 0.04 | 0.76 |
| QBCO (Curvature) | 0.98 | 12.75 | **1.80** | **1.21** | 0.04 | **0.79** |
| QBCO (Length + Curvature) | 4.54 | 12.72 | 1.94 | 1.27 | 0.04 | 0.78 |

Table 4.2: Numerical comparison in the "Elbows" environment. Our piecewise Quadratic Bézier with Convex Optimization method (QBCO) has shorter length when optimizing for length, and lower max curvature when optimizing for curvature for inputs from both DSS and RRT. Using DSS as input makes the length closer to optimal in comparison to starting from RRT. Numbers in bold show the best values for that column, for each input (RRT and DSS).

| Method | Time | Length | $K_{max}$ | $K_{ave}$ | Min Clear. | Ave Clear. |
|---|---|---|---|---|---|---|
| RRT | <0.01 | 42.35 | - | - | **0.59** | **1.20** |
| QBCO (Length) | 3.39 | **30.55** | 6.21 | 2.41 | 0.21 | 0.61 |
| QBCO (Curvature) | 0.81 | 36.02 | **1.68** | **0.85** | 0.15 | 0.78 |
| QBCO (Length + Curvature) | 3.14 | 35.71 | 1.84 | 0.93 | 0.17 | 0.77 |
| DSS | <0.01 | 35.71 | - | - | 0.17 | 0.77 |
| QBCO (Length) | 0.75 | **29.16** | 5.37 | 2.01 | **0.45** | 0.60 |
| QBCO (Curvature) | 0.71 | 33.73 | **1.72** | **1.00** | 0.19 | **0.79** |
| QBCO (Length + Curvature) | 1.77 | 33.42 | 3.14 | 0.74 | 0.22 | 0.78 |

Table 4.3: Numerical comparison in the "Maze" environment. Our piecewise Quadratic Bézier with Convex Optimization method (QBCO) performs well in difficult environments, decreasing both path length and max curvature in reasonable computation time. Numbers in bold show the best values for that column, for each input (RRT and DSS).

to the input path before applying QBCO, so that QBCO can reach convergence faster.

### 4.5.2 Experiments

We have tested and evaluated the effectiveness of our piecewise Quadratic Bézier with Convex Optimization method (QBCO) in various environments: maze,

mixed, inter-locked, simple, office, elbow, and regular; which were also used in previous path planning work [83]. These environments include convex and non-convex obstacles varying in size and placement.

We have used the Gurobi optimization package to solve our convex optimization problem. We have tested QBCO with both RRT and DSS paths as input, and with different weights for the objective function. We propose results optimizing only path length (by setting the weight for curvature to be zero), only around max curvature (by setting the length weight to zero), and optimizing both length and curvature with equal weights. For all tests, the weight of clearance in the objective function was left at 1, in order to ensure our paths were always collision-free. These weights can be easily changed through the user-interface we have built for the testing application.

We have utilized common metrics found in the literature: computation time, path length, maximum curvature, maximum curvature averaged over every curve in the path, minimum clearance, and minimum clearance averaged over every curve in the path. These measurements best capture the features of paths which are the focus of path optimization literature.

### 4.5.3 Discussion

Tables 4.2 and 4.3 display how effective our approach is at reducing path length and minimizing curvature in two different environments. In both of these tables, the columns for curvature show a dash symbol for RRT and DSS since those methods by themselves are piecewise linear and therefore no curvature control. The best values for each column and type of initial path (RRT or DSS) are shown in bold. The shortest length path is consistently found with QBCO prioritizing length. The path with the smallest maximum curvature is consistently found with QBCO prioritizing curvature. QBCO with equal weight for maximum curvature and length provides a middle ground in both of these metrics while maintaining the largest clearance among each method. Therefore, our method is responsive to the user-specified optimization weights.

The number of iterations of QBCO before convergence was typically less than

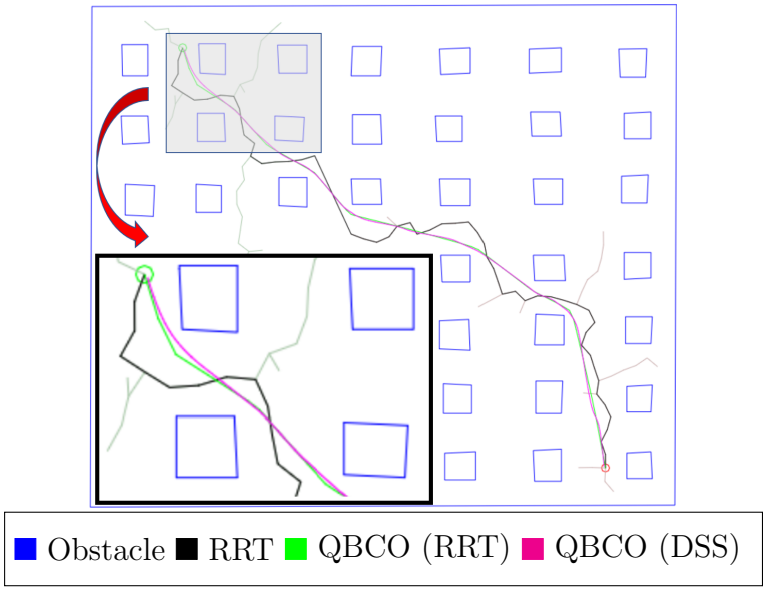


Figure 4.4: *Results in the "Regular" environment obtained from QBCO starting from RRT, versus starting from DSS, both prioritizing curvature. The path starting from DSS is smoother and shorter, since DSS allows QBCO to converge closer to the optimal solution for the same amount of iterations. The zoomed-in section shows the differences between the final paths.*

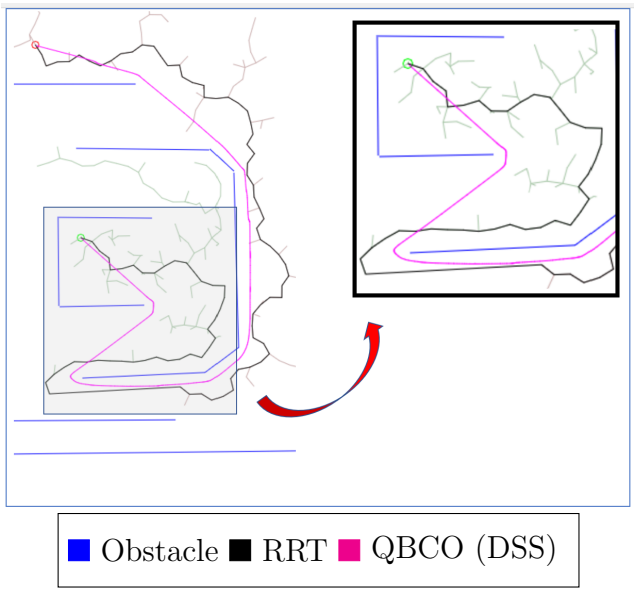


Figure 4.5: *QBCO employing DSS input in the "Inter-locked" environment with* $c_L = 0.80$ *and* $c_{K_{max}} = 0.20$*. The zoomed-in region shows that the path is straighter but has higher curvature near obstacles.*

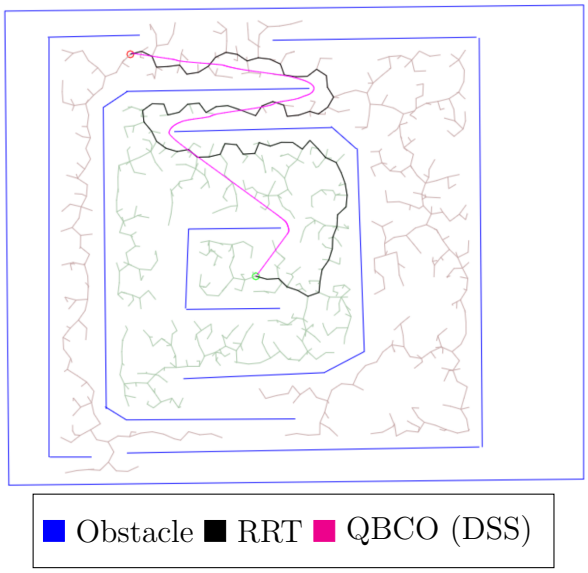


Figure 4.6: *QBCO using DSS input in the "Inter-locked" environment with* $c_L = 0.50$*, and* $c_{K_{max}} = 0.50$*. The result is balanced in terms of path smoothing and shortness.*

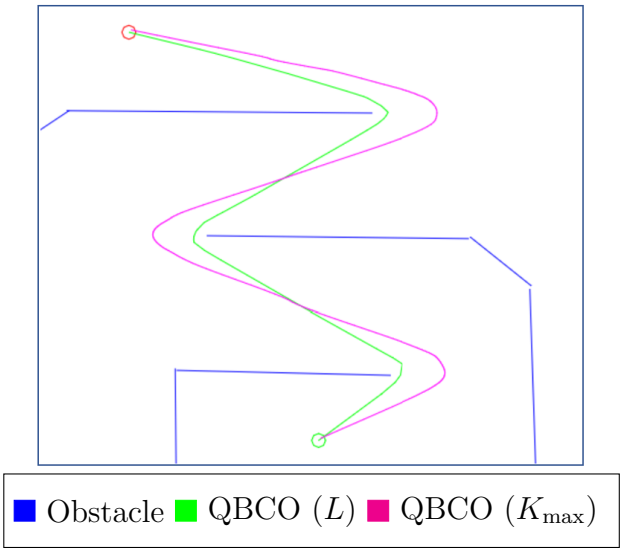


Figure 4.7: *Comparative analysis of QBCO performance between prioritizing curvature vs prioritizing length. Scenario 1 (*$c_L = 0.75$*,* $c_{K_{max}} = 0.25$*) prioritizes path length, resulting in a shorter but higher curvature path. Scenario 2 (*$c_L = 0.25$*,* $c_{K_{max}} = 0.75$*) prioritizes curvature, leading to a smoother but longer path.*

10. The running time for the convex optimization solver from Gurobi significantly differs based on the constraints and objective function imposed. As a result, the

computation times for different methods can differ significantly. The values shown in our tables are averaged over 50 runs using the same start and end points.

Figure 4.4 compares the results obtained by starting from RRT or DSS as the initial piecewise linear path for QBCO. The paths are very similar, but starting from DSS allows us to achieve a better result, in this case in terms of both path length and curvature. Figure 4.5 shows a scenario where length is prioritized. The weight for length is 0.80, while the weight for curvature is 0.20. Thus we can see the path is shorter, but may have sharper corners than if the objective function prioritized curvature.

Figure 4.6 is a more balanced example with 0.50 for the length weight, and 0.50 for the curvature weight. The length of the path is reduced, but not too much at the expense of curvature.

Figure 4.7 displays the performance of QBCO with respect to an RRT input path. Two different optimization scenarios were considered, one prioritizing length ($c_L = 0.75$ and $c_{K_{\max}} = 0.25$) and the other prioritizing curvature ($c_L = 0.25$, $c_{K_{\max}} = 0.75$), when optimizing a corner. The results reveal that when prioritizing length, the path becomes shorter but exhibits higher curvature, as illustrated in the image. In contrast, when prioritizing curvature, the path becomes smoother but longer in length.

### 4.5.4 Curvature Control

Figure 4.8 shows the max curvature of a quadratic Bézier curve when varying the x and y coordinate of one of the control points of that curve. The curvature is relatively flat and planar, except at the region of discontinuity. Therefore, as long as the control points are constrained to not reach the discontinuous region of this surface, our linear approximation of maximum curvature is a very close to the true value.

### 4.5.5 Conclusion

Robots and other real-world agents have constraints on their dynamic properties, such as a limit to how quickly they can accelerate or how fast their maximum velocity is. Therefore, we have an interest in generating paths with a limit on the their maximum curvature, since curvature is directly related to angular acceleration at a given velocity. Our proposed method directly addresses such cases by minimizing the maximum curvature of the path being optimized.

Some methods for path planning using Bézier curves provide $C^2$ continuity. When using exclusively quadratic Bézier curves, this is not possible. Quadratic Bézier curves are defined by three control points which gives each curve three degree of freedom. Enforcing each of $C^0$, $C^1$, and $C^2$ continuity requires one equality constraint in the optimization model for every curve. Therefore, enforcing $C^0$, $C^1$, and $C^2$ continuity on a series of quadratic Bézier curves represents a system of linear equations with three equations and three unknowns for each curve. Such a linear system has a unique solution if it exists, which is not suitable for convex optimization. While relying on a quadratic formulation gives us a simplified way to incorporate the curvature term, as future work we plan to extend our method to a piecewise cubic representation.

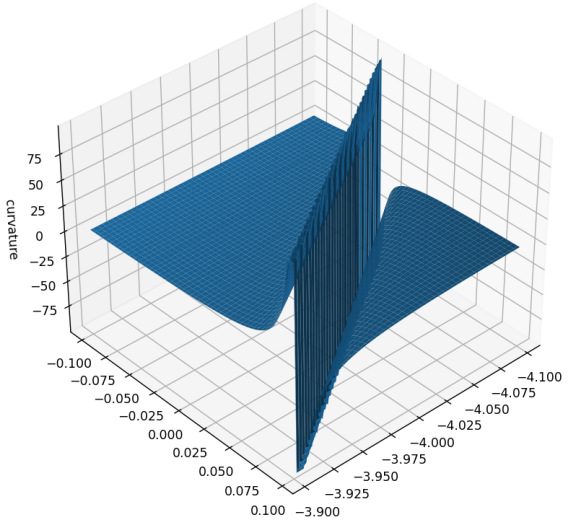


Figure 4.8: *Illustration of the maximum curvature when varying a control point of a Bézier curve. The vertical axis is curvature while the horizontal plan represents the range of x and y values the control point may take. The chasm down the middle of the figure is a result of an infinite discontinuity at that set of x-y values.*

# Chapter 5

# Multi-Objective Path Optimization for Sets of Lanes in Cluttered Environments

## 5.1 Introduction

Path computation is a crucial process in robotics and computer graphics. It is important for many applications, from autonomous vehicles to autonomous entities in computer games. In its most common form, path computation is finding a path for an agent from its initial configuration to a given goal configuration, without colliding with any declared obstacles in the environment. The path computation problem can be however defined to address different types of constraints, such as clearance from obstacles and maximum curvature. Additionally, path computation can be defined for higher dimensional problems, such as for addressing robotic manipulation [87] problems. In this chapter, I address this problem in 2D and I approach it in two phases: first, one or more paths are computed using a path planning method, and then, a path optimization phase is proposed to produce the final result addressing given constraints. The goal of this chapter is to address these steps for the case of multiple non-crossing paths which are called lanes.

For the planning phase, this chapter presents modifications to the Rapidly-

exploring Random Tree (RRT) sampling-based planner, such that sets of lanes between goal regions in an environment can be generated. Given the sampling-based nature of the method, the produced paths are often highly irregular and they require an optimization procedure. Figure 2.1 summarizes how trees and paths are generated by this method. We can see the tree growing from the initial configuration $q_{\text{init}}$ towards the goal configuration $q_{\text{goal}}$. Please refer to Section 2.1 for more details.

Optimizing paths for multiple agents navigating a shared environment presents a significant challenge. Multi-agent path optimization plays a critical rule in scenarios where autonomous entities, such as drones, robots, or virtual agents, collectively traverse from distinct starting points to designated destinations. Common objectives include the facilitation of coordinated movements, ensuring avoidance of collisions, and the maximization of overall efficiency.

Many researchers have studied path smoothing and optimization methods. In our current study, we specifically align with the domain of optimization methods, where we use a convex optimization-based approach for our proposed model.

Our method is applied to the paths represented as piecewise Bézier Curves [90]. Splines, including Bézier curves, are widely favored for representing paths due to their popularity in various applications. [59, 63, 65, 90], since they are parameterized and, in comparison to other polynomial interpolation approaches, Runge's phenomenon can be avoided for higher degrees. B-spline curves also represent a popular approach for Spline-based path or trajectory representation [28, 77, 78]. We rely on Bézier curves because of their simpler formulation and suitability for integration into our optimization framework.

We introduce an optimization method for multiple agents called Piecewise Quadratic Bézier Curve Multi-agent Convex Optimization (MCO) that enables users to customize the terms in the objective function to control clearance, length, and curvature. In order to test our method, we present a multi-agent formulation of Rapidly-exploring Random Trees (RRT) with lazy collision checking to produce sets of lanes to be optimized. We outline a series of optimization constraints that guarantee $C^1$ continuity, and as well define a objective function formulated to

address length, curvature, and clearance.

We provide several benchmarks to show the results of the method, and also to present comparisons against the Multi Deterministic Shortcut-based Smoothing (MDSS) method presented in Chapter 3. The presented method outperforms the compared methods in speed of computation, and gives users the flexibility to customize the properties of the obtained paths by tuning the weights in the multi-objective function.

## 5.2 Related Work

We categorize Path-planning algorithms for multiple agents into Geometric and Numerical Optimization Methods. Geometric methods follow classical approaches like Artificial Potential Field, Sampling-based Approaches, and Graph-Based Approaches, while Numerical optimization methods include heuristic algorithms and AI-based approaches. In the literature review section of this thesis, I review papers in the areas of geometric-based and numerical optimization-based methods. However, in this specific section, my focus is primarily on the papers that I intend to compare with the work presented in this chapter.

### 5.2.1 Geometric Optimization Methods

In [18], the authors propose a geometric method of minimizing the curvature of a quadratic Bézier curve inside a bounding tetragon. This method, like the author's other proposed methods, rely on very strict environmental constraints and rely on having a high-quality initial path with equal vertex spacing to get the best results. Therefore this method and the others the authors proposed have very limited curvature control and no user-specified conditions or preference which guide path optimization. We decided to focus on optimization in this paper in order to provide a more general purpose optimization algorithm which functions in multiple passes on initial paths of any condition. Their method results are strongly dependent on the initial paths and it is only capable of moving the Bézier path small amount and only one time.

Huang et al [44] propose a method which uses a generalized Voronoi diagram (GVD) to divide free space into regions based on each robot's path-priority order, which is a predefined sequence of robots that determines who has the right to move first in case of conflict. In contrast with [44], which focuses solely on comparisons for average trajectory length and success rate, our formulation introduces a customizable objective function, allowing users to adjust weights for not only length, but also curvature and clearance terms. [44] does not discuss the concept of smoothness in trajectory planning. However, our Work emphasizes achieving $C^1$ continuity for smooth transitions in multi-agent systems with a target minimum clearance to obstacles. Our formulation provides explicit customization options, ensuring transparency in determining 'better' navigation points. Additionally, our approach considers multiple paths simultaneously for optimal shortcut point selection, offering a more comprehensive exploration strategy compared to [44].

Chen et al [13] propose a novel method for path planning of multiple agents in an intelligent warehouse, using an Artificial Potential Function (APF) and wall-following strategy. Utilizing simulated forces, the Artificial Potential Field approach directs agent movements by shaping their interactions with the environment's potential field. The paper aims to solve the problems of local minima, non-reachable target, collision and traffic jams that may occur in multi-robot systems.

The distinction between our approaches and [96] is marked. We evaluate our algorithm's versatility by testing it against various multi-agent path optimization techniques, considering a diverse range of complex environments with obstacles of varying shapes and sizes. In contrast, their approach falls short in its experimental methodology. Their work lacks a comprehensive analysis of computational complexity and convergence rate, critical for understanding how well the algorithm performs. Furthermore, the absence of comparisons with other state-of-the-art methods and the oversight of communication and coordination in [96] challenges among robots limit the practicality and robustness of their proposed approach. Our work addresses these gaps, providing a more thorough examination of algorithmic performance, applicability, and generalization.

### 5.2.2 Numerical Optimization Methods

Zhang et al [102] propose a method of curvature control for single piecewise polynomial curve paths which linearizes the expression for the curvature of the path for the sake of optimization. While this method includes curvature, obstacle clearance, and dynamic considerations such as jerk in its optimization problem, the authors notably do not seek to improve the overall length of the final path. At the same time, the method of finding an initial piecewise polynomial curve path is not discussed, nor how one may convert from a piecewise linear path to a polynomial path compatible with their representation. As a result, their paths tend to increase in length with iterations of optimization and do not show a clear way for the user to specialize or provide input paths.

Dmitri et al present an algorithm [27] for generating smooth paths for self-driving vehicles, considering real-time obstacles detected by the robot's sensors. This work involves two key steps: In the initial stage, a variation of the $A*$ search algorithm is applied to the vehicle's 3D kinematic state space. However, a modified state-update rule is employed to capture the continuous state of the vehicle within the discrete nodes of $A*$, ensuring the path's kinematic feasibility. Subsequently, the second step focuses on improving the solution's quality through numeric non-linear optimization, ultimately achieving a local optimum.

This work lacks a comparison with state-of-the-art methods, and concentrates solely on single-agent path optimization. Unlike our work, the method proposed in [27] does not use piecewise-curved paths but only piecewise-linear paths, while approximating curvature from the angle between path segments. This limits the researchers capacity to accurately optimize around the curvature of the path.

Siedentop et al [89] propose an automated parking system that navigates without additional restrictions. It uses a lattice grid search and optimization for a desirable solution, with Dubins Curves for grid search edges. However, it introduces constraints like curvature and orientation cost, and lacks a mechanism for user-defined curvature preferences. The shapes of the optimized paths are limited by their choice of path framework, which cannot guarantee $C^1$ or curvature continuity. The authors make an approximation of the maximum curvature of a

path segment for their system of constraints which does not allow the full range of vehicle motion. Our work, in contrast, allows users to adjust weights for trajectory length, curvature, and clearance. The previous work also lacks analysis of computational complexity and convergence rate, and their claims about the convexity of the objective function and the independence of path length from grid resolution lack verification. Our manuscript addresses these gaps and provides a comprehensive analysis of our proposed approach. Their claims about the convexity of the objective function and the independence of path length from grid resolution lack mathematical and statistical verification. Our manuscript enhances the field by addressing these gaps and providing a comprehensive analysis of our proposed approach.

Zhanna et al [32] propose a new algorithm for multi-agent path planning in unknown environments, where the robots use their sensors to detect obstacles and other robots. The algorithm considers two objectives: finding the safest and the shortest path for each robot. The paper uses Voronoi Diagrams to find the safest path, which is the geometric location that is farthest from all obstacles. This paper offers no algorithmic focus on optimizing clearance and curvature control. However, our Approach achieves $C^1$ continuity using the Piecewise Quadratic Bézier Curve Multi-robot Convex Optimization (MCO) algorithm, ensuring smooth trajectories.

The approach discussed in [32] unifies the generation of start and goal points but lack options for specifying regions or distributing points across the environment. In contrast, our approach offers various options, including uniform and region-specific point selection, facilitating controlled evaluations in specific scenarios. Our approach addresses these gaps, providing a thorough examination of algorithmic performance, applicability, and generalization.

Ye et al [98] propose method for the trajectory following using curvature control. This paper shows the importance of understanding dynamic properties of the vehicle in following a predetermined path. Our proposed method are therefore more unique in the round of path optimization rather than trajectory following. Because the literature on using curvature control in path optimization is more sparse.

Li et al [64] present a method for optimal trajectory design for multi-UAV systems using particle swarm optimization (PSO). It generates smooth, collision-free paths and outperforms existing algorithms in convergence and accuracy. While they focus on energy usage and flight risk, our work also considers path length and trajectory smoothness for a more comprehensive UAV path design process. One notable distinction lies in the title of this paper [64], claiming "Optimal trajectory UAV path design based on Bézier curve". However, our critical analysis reveals a lack of proof or explanations regarding how their approach is truly rooted in Bézier curves, let alone specifying the degree of the curve employed. In contrast, our work provides a detailed and transparent explanation of our utilization of Bézier curves, elucidating precisely how we control the curvature of the paths. Additionally, authors in [64] employ Particle Swarm Optimization with Fuzzy Mutation (PSO-FM2) for path optimization. However, we raise concerns about the known limitations of PSO, such as premature convergence and susceptibility to local optima. In contrast, we advocate for the exploration of more robust optimization techniques to enhance the overall performance and reliability of the trajectory optimization method.

Huang et al [45] propose an improved ant colony optimization (ACO) algorithm for the multi-agent path finding (MAPF) problem, addressing limitations of the standard ACO. However, it does not address smoothness, curvature control, or optimize clearance and curvature. Our approach, the Piecewise Quadratic Bézier Curve Multi-agent Convex Optimization (MCO) algorithm, achieves $C1$ continuity and checks additional matrices for average and sharpest angles. The paper also lacks a rigorous mathematical formulation of the MAPF problem and the IACO algorithm, and does not consider scalability and robustness [45]. Our approach uses a well-defined mathematical formulation and explores diverse environments..

## 5.3 Method

### 5.3.1 Problem Description

#### Problem Statement

Given a set of non-intersecting piecewise-linear paths $Q$, and a set of polygonal obstacles $O$, we must optimize the paths around the user-specified objective function while avoiding intersections between paths and intersections of paths with obstacles. Let the set of initial paths be $Q = \{q_1, q_2, ..., q_Z\}$, where $Z$ is the number of path that needs to be optimized. Each path $q_i$ is an ordered set of vertices $q_i = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_N\}$, where each vertex is a point in 2D space, $\mathbf{v}_i \in \mathbb{R}^2$. The obstacles are a set of polygons $O = \{O_1, O_2, ...\}$, where each polygon is a closed set of vertices $O_m = \{\mathbf{o}_1, \mathbf{o}_2, ...\}$.

Our proposed method first transforms the input paths to an ordered set of Quadratic Bézier paths $Q' = \{q'_1, q'_2, ..., q'_Z\}$. Each piecewise-quadratic Bézier curve consists of a set of Bézier curves $q'_z = \{B_1, B_2, ..., \}$. Each quadratic Bézier curve can be defined by its three control points,

$$B_i = \{(1-t)^2\mathbf{p}_{i,0} + 2t(1-t)\mathbf{p}_{i,1} + t^2\mathbf{p}_{i,2} \\ \forall t \in (0,1) \mid \mathbf{p}_{i,0}, \mathbf{p}_{i,1}, \mathbf{p}_{i,2} \in \mathbb{R}^2\}$$

Our method computes and improves the positions of the control points for each path $q'_i$, each of which define a continuous collision-free path. In this section, we employ the notation $B(t)$ to refer to a given quadratic Bézier path evaluated at $t$.

Our optimization criteria are curvature, clearance, and path length. We enforce $C^1$ continuity for each Bézier curve while minimizing the sum of the maximum curvature along the path, the distance between the path and obstacles, and the length of the path, subject to user-specified coefficients.

#### Generating Piecewise Linear Paths

We introduce an extension of the popular Rapidly-Exploring Random Trees (RRT) method for multi-agent systems, which we call Simultaneous RRT (SRRT).

In this technique, we create a pair of trees for each path we would like to create, one rooted in the start point of the path, and the other rooted at the goal point.

We employ a collision avoidance approach in this option, allowing each path to explore freely while preventing collisions between the multiple trees. We let the trees grow and we add branches without checking collision with other trees. Then, only when a solution is found for tree $i$, then we check for collisions between path $i$ all other trees. We then attempt to reconnect any orphaned nodes of tree $j$, if such branches are collision-free.

Alternative methods for creating initial piecewise-linear paths were explored, but found to be not efficient or unlikely to succeed in finding a path from each start to each goal point. These include generating one path at a time according to some heuristic priority and searching for paths in a high-dimensional space which is then mapped onto a variable number of paths in 2 dimensions (e.g. a tree in 6 dimensional configuration space may be mapped onto 3 paths in 2D space).

To generate start and goal points between which we can make path using RRT, we allow the user to easily designate start and goal polygons. Start points are randomly distributed in the start polygon and are randomly assigned to the goal points in the goal polygon. Concentrating start and goal points in the same zones allows for a controlled evaluation, emphasizing the algorithm's ability to coordinate paths within specific regions.

### Transforming to Piecewise Bézier Paths

To create a smooth (e.g. $C^1$ continuous) path from a piecewise linear path, we devise a simple curve fitting technique which involves creating one quadratic Bézier curve for each vertex of the input polygonal path, for each path $q_i$ where $\forall i \in Z$. For each vertex $\mathbf{v}_i \in Q_i, i \neq 1, N$, we define the control points for curve $B_i = \{\mathbf{p}_{i,0}, \mathbf{p}_{i,1}, \mathbf{p}_{i,2}\}$ in $P'$ as:

- $\mathbf{p}_{i,0} = 0.5(\mathbf{v}_{i-1} + \mathbf{v}_i)$,
- $\mathbf{p}_{i,1} = \mathbf{v}_i$,
- $\mathbf{p}_{i,2} = 0.5(\mathbf{v}_i + \mathbf{v}_{i+1})$.

Thus, $\mathbf{p}_{i,0}$ is the midpoint between $\mathbf{v}_i$ and the previous vertex, while $\mathbf{p}_{i,2}$ is the midpoint between $\mathbf{v}_i$ and the next vertex. For the first vertex, $\mathbf{p}_{0,0} = \mathbf{v}_0$, and for the last vertex, $\mathbf{p}_{N-1,2} = \mathbf{v}_N$.

If a collision is found for a given curve, the first and last control points of the curve are moved to the mid-points between their original positions and vertex $\mathbf{v}_i$, effectively making the curve to be closer to the input polygonal path at vertex $\mathbf{v}_i$. When this binary subdivision is performed, two new Bézier curves forming straight segments are added to connect the created gap at the beginning and end of the modified curve. This process can be repeated recursively in order to move the Bézier curve arbitrarily close to the linear path, ensuring that no collisions are introduced when converting the input path to the piecewise Bézier representation.

### Optimization Variables

Let $\{\mathbf{m}_{i,0}, \mathbf{m}_{i,1}, \mathbf{m}_{i,2}\} = B_{m,i} \in M, \forall i$, be variables for the optimization model $M$, where $\mathbf{m}_{i,j} = (x_{m_{i,j}}, y_{m_{i,j}})$. Each $\mathbf{m}_{i,j}$ corresponds to $\mathbf{p}_{i,j} \in q_i'$. We can evaluate the model variables just like we do with the control points:

$$B_{m,i}(t) = (1-t)^2\mathbf{m}_{i,0} + 2t(1-t)\mathbf{m}_1 + t^2\mathbf{m}_{i,2}. \tag{5.1}$$

### Optimization Constraints

We optimize the model for each path $q_i$, $\forall i \in Z$ subject to the following constraints:

$$B_i(t=1) = B_{i+1}(t=0) \Rightarrow \mathbf{m}_{i,2} = \mathbf{m}_{i+1,0}, \tag{5.2}$$

$$B_i'(t=1) = B_{i+1}'(t=0)$$

$$\Rightarrow \mathbf{m}_{i,2} - \mathbf{m}_{i,1} - \mathbf{m}_{i+1,1} + 2\mathbf{m}_{i+1,0} = 0, \tag{5.3}$$

$$\|\mathbf{m}_{0,0} - \mathbf{v}_1\| < \epsilon, \tag{5.4}$$

$$\|\mathbf{m}_{N-1,2} - \mathbf{v}_N\| < \epsilon, \tag{5.5}$$

$$\|\mathbf{m}^* - \mathbf{c}\| < r_c. \tag{5.6}$$

$C^0$ **Continuity**: This constraint is expressed in equation 5.2. $C^0$ continuity

means two adjacent (ordered one after the other) curves for each Bézier path $q'_i$, $\forall i \in Z$ share an endpoint.

$C^1$ **Continuity**: Equation 5.3 establishes the condition for $C^1$ continuity. This constraint means the tangent vectors of two adjacent curves for each Bézier path $q'_i$, must be equal at their shared endpoint. The resulting path therefore has smooth transitions between each consecutive pair of Bézier curves, with a continuous first derivative.

**Endpoints Constraint**: Constraints 5.4 and 5.5 are related to the start and end points of the path. In our problem statement, we stated that the optimized Bézier path $q'_i$, should maintain the same start and endpoint as the initial linear path $q_i$. If we enforce this exactly by constraining $\mathbf{m}_{0,0} = \mathbf{p}_{0,0}$ we would make the optimization model infeasible; therefore, we accept a small difference, $\epsilon$, in the position of the start and end of the path.

**Distance Constraint**: Equation 5.6 prevents collisions between the path and obstacles while also serving as the clearance objective. We identify a collision-free region around the Bézier path $q'_i$ by a collection of collision-free disks. The model $M$ is constrained to move the path only within the collision-free region.

The center points of the disks are placed along the path, and the radii of the disks are set to be the distance between each path center and its closest obstacle.

To define these disks, we choose a set equally-spaced points along the path. For each chosen point, $\mathbf{p}^*$, we find the closest point $\mathbf{o}_{min}$, among the obstacles to $\mathbf{p}^*$:

$$\mathbf{o}_{min} = \underset{\mathbf{o} \in O_j, O_j \in O}{\arg\min} \|\mathbf{o} - \mathbf{p}^*\|,$$

with corresponding distance being:

$$d_{min} = \|\mathbf{o}_{min} - \mathbf{p}^*\|.$$

We thus take the center of the collision-free disk corresponding to the chosen point to be equal to point $\mathbf{c} = \mathbf{p}^*$, with radius $r_c = d_{min} - \delta$, where $\delta$ is the desired clearance between the path and obstacles. The optimization model expression

corresponding to $\mathbf{p}^*$ is $\mathbf{m}^*$. Precisely:

$$\mathbf{p}^* = B_i(t^*) = (t^*)^2\mathbf{p}_{i,2} + 2t^*(1-t^*)\mathbf{p}_{i,1} + (1-t^*)^2\mathbf{p}_{i,0}$$
$$\Rightarrow \mathbf{m}^* = (t^*)^2\mathbf{m}_{i,2} + 2t^*(1-t^*)\mathbf{m}_{i,1} + (1-t^*)^2\mathbf{m}_{i,0}.$$

This disk is guaranteed to be collision-free, since the distance between $\mathbf{p}^*$ and $\mathbf{m}^*$ is constrained to be less than the distance between $\mathbf{p}^*$ and the nearest obstacle to $\mathbf{p}^*$. This constraint is illustrated in Figure 5.1. Since our environments and paths are 2-dimensional, we can further express constraint 5.6 as:

There exist several methods for calculating the distance between Bézier curves, such as those based on culling [11]. Using such methods may improve the running time of our proposed algorithm, but we leave this for future work.

$$(x_{m^*} - x_c)^2 + (y_{m^*} - y_c)^2 < r_c^2. \tag{5.7}$$

This distance constraint has the added benefit of improving the accuracy of our approximation for the curvature of each Bézier curve, as discussed in the next section.

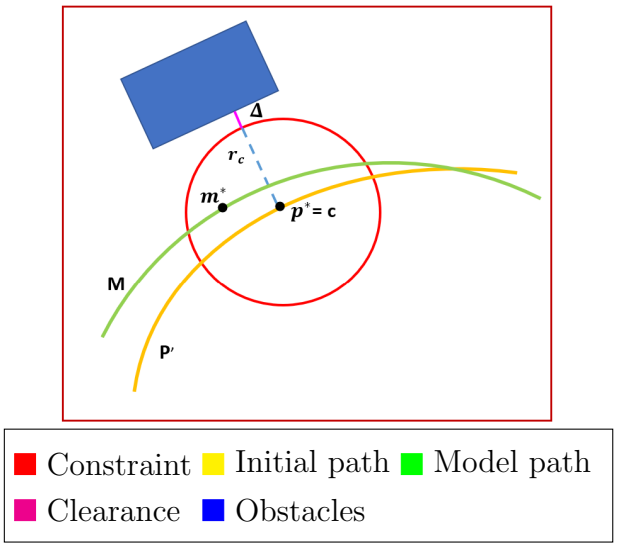


Figure 5.1: *Visualization of the Distance constraint*

**Multi-Objective Optimization Function**

Since there is more than one independent quantity to optimize, our problem requires multi-objective optimization. We assign a weight (or coefficient) to each objective term to indicate its relative importance in the model. we formulate our objective function considering maximum curvature $K_{\max z}$, clearance $\Delta_z$, and length $L_z$ for each path $q_i'$, as below.

$$\text{Minimize} \quad \sum_{i=0}^{Z} \sum_{i=0}^{n-1} (c_K \hat{K}_{\max_i} + c_\Delta \Delta_i + c_L \hat{L}_i). \tag{5.8}$$

The maximum curvature along a single quadratic Bézier curve can be calculated with:

$$K_{\max} = \frac{\|\mathbf{p}_2 - 2\mathbf{p}_1 + \mathbf{p}_0\|}{2\|(\mathbf{p}_1 - \mathbf{p}_0)\| \cdot \|(\mathbf{p}_2 - \mathbf{p}_1)\|}. \tag{5.9}$$

This formula however is not suitable for optimization packages, since it is non-linear, non-convex, and discontinuous. Therefore, we approximate the max curvature using a first-degree Taylor Expansion, therefore linearizing the equation. The linearization involves taking derivatives of $K_{\max}$ with respect to each of the optimization variables, evaluated at the corresponding control points of $q_i'$, $\forall i \in Z$ , which we take as our *base*. We apply the process of linearizing a function $y = f(x)$ with:

$$\hat{y} = (x - x_{base}) \left.\frac{df}{dx}\right|_{x=x_{base}} + f(x_{base}). \tag{5.10}$$

Where $\hat{y}$ is the linear approximation for $y$. Applying this process to $K_{\max}$ results in:

$$\begin{aligned} \hat{K}_{\max} = {} & (x_0 - x_{0,base}) \left.\frac{dK_{\max}}{dx_0}\right|_{x=x_{0,base}} \\ & + (y_0 - y_{0,base}) \left.\frac{dK_{\max}}{dy_0}\right|_{y=y_{0,base}} \\ & + ... \\ & + K_{\max}(x_{0,base}, y_{0,base}, ..., y_{2,base}). \end{aligned} \tag{5.11}$$

Where $\hat{K}_{\max}$ is the linear approximation for curvature. Expressions for each derivative of $K_{\max}$ can be found in the Appendix. This linearized version of max

curvature is implemented in the objective function, substituting $\mathbf{p}_i$ with the corresponding optimization variable $\mathbf{m}_i$. We sum the max curvature of every curve along the path in order to minimize the maximum curvature throughout the path, rather than myopically focusing on only one curve at a time.

We represent the length of the entire piecewise Bézier curve as $L = \sum L_i$, where $L_i$ is the length of each curve $B_i$. A quadratic Bézier curve $B_i(t)$ is defined on the range $t \in (0, 1)$. Therefore, we can express the arc length of this curve with:

$$L_i = \int_0^1 \left\| \frac{dB_i}{dt} \right\| dt. \tag{5.12}$$

A method for calculating the exact arc length for a quadratic Bézier curve is outlined in the Appendix Section 7.

*Clearance* in our convex optimization problem refers to the minimum distance between the path generated by the optimization algorithm and any obstacles in the environment. To ensure that the system is safe, it is important to consider clearance in the optimization problem.

Clearance is expressed in the objective function as a pseudo indicator function where control points with adequate clearance have low cost, and control points outside of the collision-free region have very high cost.

### 5.3.2 Proposed Method

#### User-specified termination conditions for convex optimization

For all convex optimization methods discussed in this chapter, we consider three options for their termination conditions. The user may choose either a time limit, a round limit, or convergence. Using the round limit condition, convex optimization continues for the specified number of rounds. During each round of convex optimization, the control points which define the path are allowed to move up to a configurable distance. Using the time limit condition, optimization will continue until the specified time has elapsed, regardless of how many rounds are run. Finally, when using the convergence condition, optimization continues until the update to the path on the most recent optimization round is sufficiently small.

We define the size of the update to the path as the sum of the distances between the control points in the input path(s) and the control points in the output of convex optimization. The threshold defining convergence is also user-configurable.

### Use of Single-agent Convex Optimization

The primary contribution of this publication is the proposal of a method for multi-agent planning which relies on our previous work in single-agent convex optimization [56]. The pseudocode for this algorithm is repeated here 6. This algorithm implements the optimization variables, constraints, and objective function described in section 5.3.1, and is used iteratively so solve multi-agent systems.

**Algorithm 6:** Single Agent Convex

**Data:** Piecewise Bézier path $q'$, Obstacles $O$

$M \leftarrow$ CreateConvexModel()

**while** $q'$ *not converged* **do**

- AddVarsFromBezier($M$, $q'$)
- AddObjLinearCurvature($M$, $q'$)
- AddObjApproxLength($M$, $q'$)
- AddEndConstraints($M$, $q'$)
- AddC1Constraints($M$, $q'$)
- AddDistConstraints($M$, $q'$, $O$)
- Optimize($M$)
- $q' \leftarrow$ ExtractPath($M$)

**end**

### Priority Ordering

In this method, the optimization process alternates between lanes, in an order determined by a priority score. We provide the user the ability to choose the relative importance of factors such as path length, distance to obstacles, and angle sharpness in the priority score. We found in our testing that a simple heuristic weighted sum of these factors gave the user sufficient flexibility to control the path

ordering while remaining intuitive. In order to ensure that every path ultimately will be optimized, we use the previous update size to the path as a global factor on the priority score as well. Therefore, paths which could not be updated much will be sorted after paths which have more opportunity to be optimized.

#### Multi-agent Interleaving Convex Optimization Method

We present Multi-agent Interleaving Convex Optimization (MICO), a robust approach to optimizing multi-agent lane systems using convex optimization techniques. It involves iterative optimization of each lane. When optimizing lane $i$, we employ the single-agent optimization technique from [56], algorithm 6, treating other lanes as obstacles with respect to lane $i$. Lanes can approach optimal configurations, even if they are temporarily prevented from doing so by the positioning of other lanes in the system.

MICO is capable of uncovering various aspects of the optimization process, including its convergence behaviors. This method is helpful when you want to make sure all the lanes are balanced or receive similar computational time during optimization.

Algorithm 7 illustrates the pseudocode for this process. In this algorithm, we consider the termination conditions on all the paths rather than only one at a time; only if convergence (for example) has been reached for all the paths does this while loop terminate. Furthermore, through the use of a priority queue, we ensure that the paths which are most in need of optimization are optimized first. The same path may be optimized multiple times in a row, but through careful design of the priority function, we ensure all the paths are eventually optimized.

### 5.3.3 Alternative Approaches

#### Multi-agent Baseline Convex Optimization

The Multi-agent Baseline Convex Optimization (MBCO) method employs a sequential and individual optimization strategy for each path lane. This involves optimizing one lane to completion before proceeding to the next, and so forth. This

**Algorithm 7:** Multi-agent Interleaving Convex Optimization (MICO)

**Data:** Priority Queue of Input paths $Q = \{q_1, q_2, ..., q_Z\}$, Obstacles $O$

**while** *Termination conditions not met on* $Q$ **do**

$q_i \leftarrow Q$.pop()

$env \leftarrow O \bigcup Q \setminus q_i$

SingleAgentConvex($q_i$, $env$)

$p \leftarrow$ priority($q_i$)

$Q$.priorityInsert($q_i$, $p$)

**end**

method represents a naïve approach to multi-agent path optimization, treating each lane as a separate component.

When optimizing lane $i$, we utilize the same single-agent optimization technique discussed in [56], while considering each other lane as an obstacle with respect to lane $i$. The quality of the optimized paths therefore depends strongly on the order in which each path is optimized.

Algorithm 8 illustrates the pseudocode for this process. In this algorithm, we consider the termination condition for one path at a time. That is, once the round limit, time limit, or convergence has been reached for path $i$, we move on to optimizing path $i + 1$.

**Algorithm 8:** Multi-agent Base-line Convex Optimization (MBCO)

**Data:** Input paths $L = \{q_1, q_2, ..., q_Z\}$, Obstacles $O$

**for** *every path* $q_i \in L$ **do**

$env \leftarrow O \bigcup L \setminus q_i$

**while** *Termination conditions not met on* $q_i$ **do**

SingleAgentConvex($q_i$, $env$)

**end**

**end**

**Multi-agent Convex Optimization Problem**

To summarize, we have outlined the following convex optimization problem $M$ for each path $q_i$, $\forall i \in Z$:

$$\text{Minimize} \quad \sum_{i=0}^{z} \sum_{i=0}^{n-1} (c_K \hat{K}_{\max_i} + c_\Delta \Delta_i + c_L \hat{L}_i) \tag{5.13}$$

Subject to:

$$\mathbf{m}_{i,2} = \mathbf{m}_{i+1,0},$$

$$\mathbf{m}_{i,2} - \mathbf{m}_{i,1} - \mathbf{m}_{i+1,1} + 2\mathbf{m}_{i+1,0} = 0,$$

$$\|\mathbf{m}_{0,0} - \mathbf{v}_1\| < \epsilon,$$

$$\|\mathbf{m}_{N-1,2} - \mathbf{v}_N\| < \epsilon,$$

$$\|\mathbf{m}^* - \mathbf{c}\| < r_c.$$

Since our objective function is an approximation of the true max curvature and length, we chose to limit the distance between $\mathbf{m}_{i,j}$ and $\mathbf{p}_{i,j}$, $\forall i, j$, in addition to the above constraints. This ensures that our approximations have small enough error to the true max curvature and length. To account for this limitation, we run the optimization program a few times, until convergence is reached. After each iteration, Bézier path $q_i'$, $\forall i \in Z$ is updated with the values calculated through convex optimization in $M$. The path is considered converged if the distance between a Bézier path $q_i'$, $\forall i \in Z$ and $M'$ is sufficiently small after an iteration. This process of running optimization multiple times also allows us to update the distance constraint before each iteration in order to precisely capture the free space available as the path changes shape. Algorithm 9 illustrates the pseudocode for this process.

**Algorithm 9:** Piecewise Quadratic Bézier Curve Multi-agent Convex Optimization (MCO)

**Data:** Input paths $Q = \{q_1, q_2, ..., q_Z\}$, Obstacles $O$
$M \leftarrow$ CreateConvexModel()
**while** *Termination conditions not met on* $Q$ **do**
  **for** *every path* $q_i \in Q$ **do**
    AddVarsFromBezier($M$, $q_i$)
    AddObjLinearCurvature($M$, $q_i$)
    AddObjApproxLength($M$, $q_i$)
    AddEndConstraints($M$, $q_i$)
    AddC1Constraints($M$, $q_i$)
    AddDistConstraints($M$, $q_i$, $O$)
  **end**
  Optimize($M$)
**end**

## 5.4 Results and Evaluation

We have applied our method to optimize different types of input paths, in several environments and using varied sets of parameters.

### 5.4.1 Experiments

We formulate our path planning problem as a convex optimization problem and solve it using Gurobi optimization package. We applied MCO, MICO, and MBCO on randomly generated input paths and varied the weights for the objective function to evaluate its performance. We experimented with MCO using different objective function weights for path length, max curvature, and clearance. We usually set the clearance weight to 1 for all tests to avoid collisions; however, we want to emphasize that it is not necessary to make the sum of the weights in the objective function equal 1. We also tested three scenarios: only minimizing length (curvature weight = 0), only minimizing curvature (length weight = 0), and

minimizing both length and curvature equally (length weight = curvature weight). We built a user-interface for the testing application that allows easy adjustment of the weights.

We used common metrics from the literature to evaluate the paths: computation time, path length, max curvature, average max curvature over all curves, min clearance, and average min clearance over all curves. These metrics correspond to the path features that are most significant for path optimization literature.

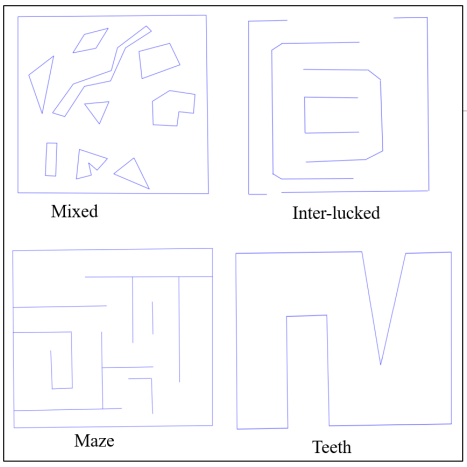


Figure 5.2: *Illustration of the environments showcases a variety of convex and non-convex obstacles, each differing in size and positions*

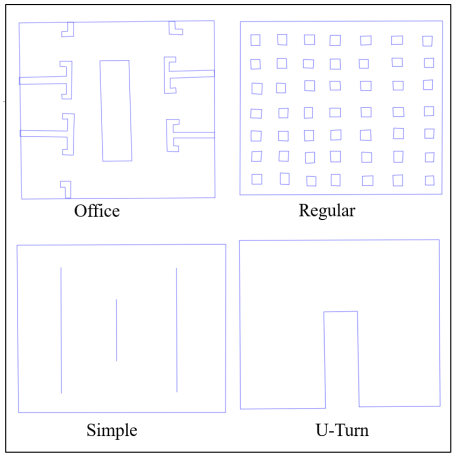


Figure 5.3: *Illustration of the environments showcases a variety of convex and non-convex obstacles, each differing in size and positions*

### 5.4.2 Discussion

Each of our convex optimization methods usually converged in less than 50 iterations. In this work, we conducted SRRT to obtain the initial piecewise linear path, and subsequently applied the Multi-agent Interleaving Convex Optimization Method (MICO), Multi-agent Baseline Convex Optimization (MBCO) and Piecewise Quadratic Bézier Curve Multi-agent Convex Optimization (MCO).

In Figures 5.2 and 5.3, various environments are represented along with their names and distinctive features.

Figure 5.4, and 5.5, present a comparative analysis of the results when comparing MICO and MBCO in the "U-Turn" environment. In Figure 5.4, it is apparent that MBCO, which optimizes one path at a time, displays sub-optimal results due to the chosen path optimization order. Notably, the prioritization of optimizing paths begins with the outer path followed by the inner path, impacting the quality of optimization achieved with MBCO. Figure 5.5 illustrates the effectiveness of MICO. This figure presents initial paths from SRRT, the path shown in pink representing the optimized path from MBCO, and the path shown in green displaying the optimized path using MICO. The paths generated by MICO not only demonstrate smoother trajectories but also create shorter paths.

Figure 5.6 shows the impact of prioritizing objective function in the "Teeth" environment using MICO. The path highlighted in green is derived by prioritizing length $c_L = 0.75$, and $c_{K_{max}} = 0.25$, while the path in red is obtained by emphasizing curvature $c_L = 0.25$, and $c_{K_{max}} = 0.75$. The results reveal that prioritizing length yields shorter paths, although the images, especially upon zooming in, demonstrate an increase in sharpness.

Figure 5.7 indicates visualizing the impact of objective function weight in the "Office" environment using MICO. The green path represents the result with weights $c_L = 0.80$, $c_{K_{max}} = 0.20$, and $c_\Delta = 0.20$, while the red path shows the optimized trajectory with weights $c_L = 0.50$, $c_{K_{max}} = 0.80$, and $c_\Delta = 0.40$. It is important to note that the sum of the weights in the objective function does not need to equal 1. The path with a higher priority for curvature is smoother and has a larger clearance compared to the path that prioritizes length, which is shorter.

Highlighting the flexibility of MICO, this visualization underscores that adjusting the weights allows for path optimization, offering a range of trade-offs in length, clearance, and smoothness.

In Figure 5.8, we present a scenario in the "Teeth" environment where the objective function prioritizes both length and curvature. Specifically, the balanced weights assigned to length and curvature are $c_L = 0.70$, and $c_{K_{max}} = 0.30$. This weighting strategy results in a path that is noticeably smoother near sharp obstacles with shorter length compared to the initial paths.

Figure 5.9 Illustrates the optimized paths of MICO when prioritizing length in the "Regular" environment. The results indicate that the paths are shorter as we place higher priority on length. Additionally, they are relatively smooth while maintaining a small clearance from obstacles, considering $c_L = 0.70$, $c_{K_{max}} = 0.30$, and $c_\Delta = 0.20$.

In Figure 5.10, we display the optimal paths generated using MICO in the "Elbow" environment. We carefully adjust the weights of the objective function for curvature, length, and clearance in the optimization process. The outcome is a refined path that is not only shorter with enhanced clearance but also demonstrates smoothness. This underscores the success of our optimization approach, which takes into consideration various factors, resulting in a path that is both efficient and fast to compute.

In Figure 5.11, we present the optimal paths found using MCO in the "Mixed" environment. The key is to balance the weight of objective function for curvature, length, and clearance during optimization. The outcome is an optimized path in terms of shorter length and smoother vertices.

As illustrated in Table 5.1, we performed a numerical comparison in the "Regular" environment setting. The evaluation includes the MBCO, MICO, and MCO methods. MICO yielded paths with less maximum curvature, and shorter overall length and larger minimum clearance to obstacles than any other method. In our experiments, MBCO typically had the shortest computation time, while MCO typically computed paths with the largest average clearance.

We performed a numerical comparison in the "Mixed" environment setting.

The evaluation includes MICO, MBCO, and MCO methods. Table 5.2 shows that MICO yielded paths with less maximum curvature, shorter overall length and computation time and larger average clearance to obstacles than any other method. In our experiments, MCO typically had the largest minimum clearance.

Table 5.3 illustrates a numerical comparison conducted in the "Teeth" environment. MICO is then applied with the specified objective function: prioritizing length, prioritizing curvature, or combining both evenly. In this experiment, when prioritizing length, we set $c_L = 0.80$, $c_{K_{max}} = 0.20$, and $c_\Delta = 0.30$. When prioritizing curvature we set $c_L = 0.30$, $c_{K_{max}} = 0.70$, and $c_\Delta = 0.30$. When using a balanced objective function, we set $c_L = 0.50$, $c_{K_{max}} = 0.50$, and $c_\Delta = 0.30$.

The choice of objective function significantly influences the quality of the optimized path. A higher coefficient for length results in a shorter final path, while a larger coefficient for curvature reduces overall curvature. Intermediate coefficients yield a combination of these effects.

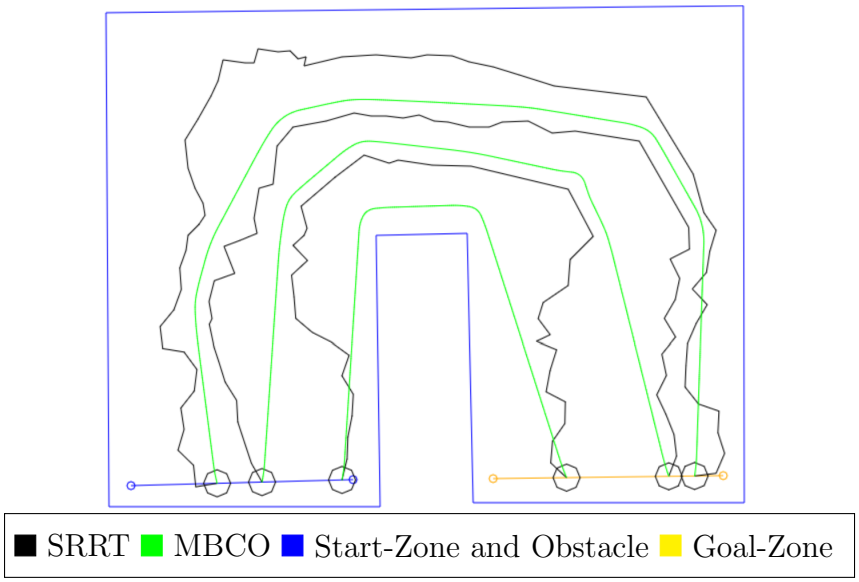


Figure 5.4: *MBCO employs SRRT input in the "U-Turn" environment. The chosen path optimization order significantly influences outcomes, resulting in sub-optimal results as paths are optimized sequentially. These paths are longer and less smooth compared to those generated by MICO.*

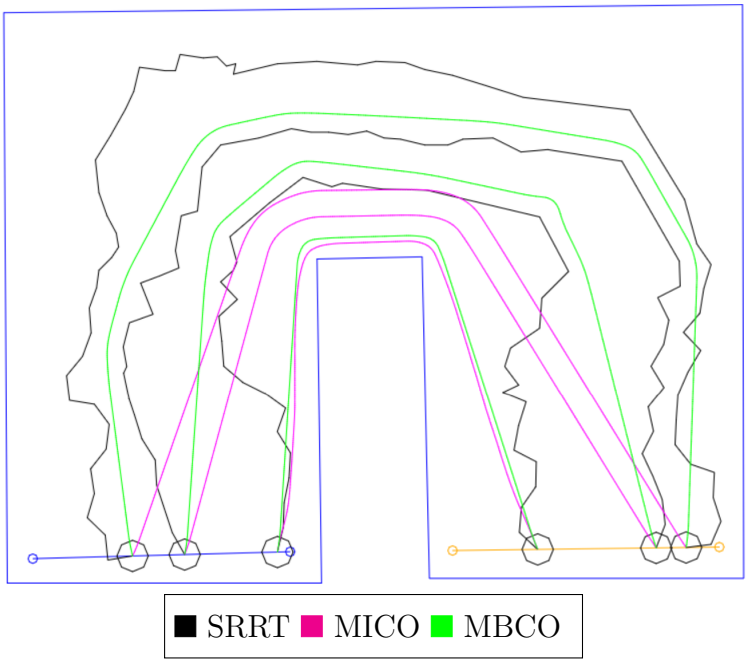


Figure 5.5: *Figure 5.5 demonstrates MICO's effectiveness in the "U-Turn" environment by comparing paths from SRRT, MBCO, and MICO. MICO produces shorter and smoother trajectories, highlighting the efficacy of MICO's performance.*

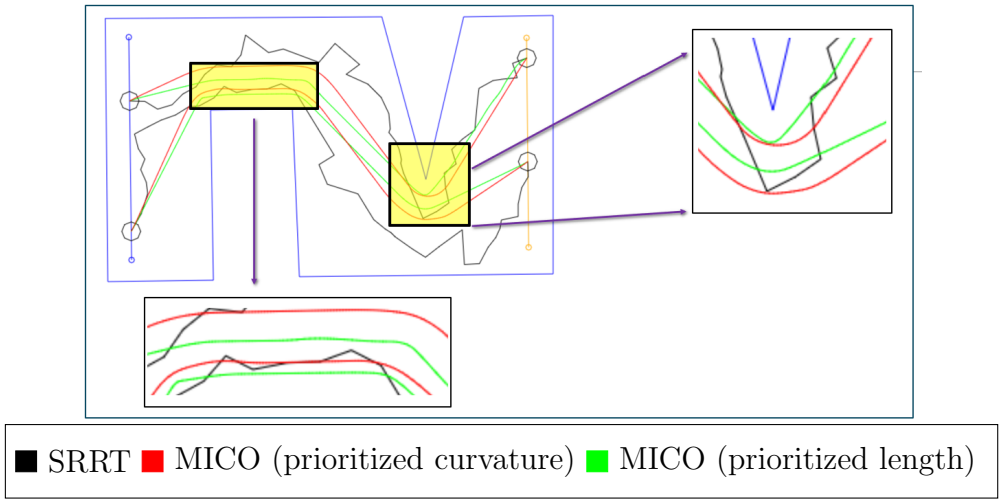


Figure 5.6: *Illustration of MICO in "Teeth" showcasing the effect of Objective function parameters, generating a green path prioritizing length* $c_L = 0.75$, $c_{K_{max}} = 0.25$ *and a red path emphasizing curvature* $c_L = 0.25$, $c_{K_{max}} = 0.75$. *Prioritizing length produces shorter paths with increased sharpness upon zooming.*

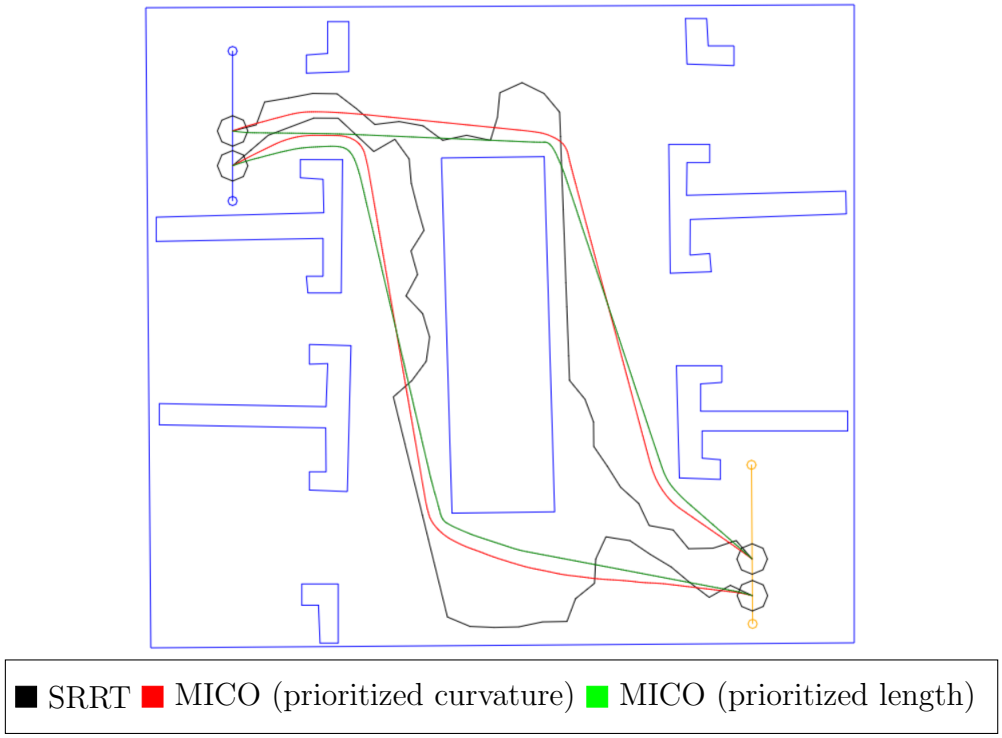


Figure 5.7: *Visualization of the comparison of objective function weight in the "Office" environment using MICO. The path shown in green shows the optimized path when* $c_L = 0.80$, $c_{K_{max}} = 0.20$, *and* $c_\Delta = 0.20$ *and the path shown in red shows the optimized path when* $c_L = 0.50$, $c_{K_{max}} = 0.80$, *and* $c_\Delta = 0.40$.

| Method | Avg T | Avg L | Max K | Avg K | Min C | Avg C |
|---|---|---|---|---|---|---|
| MBCO | **0.12** | 23.09 | 4.10 | 1.80 | 0.28 | 0.54 |
| MICO | 0.19 | **21.90** | **3.11** | **1.40** | **0.42** | 0.60 |
| MCO | 0.24 | 22.11 | 3.56 | 1.44 | 0.40 | **0.63** |

Table 5.1: Numerical comparison of multi-agent convex optimization techniques in the " Regular " environment. MICO on average computed the path with the best path length, and max curvature, min clearance. Notation: T = Time in seconds, K = Curvature, L = Length, and C = Clearance.

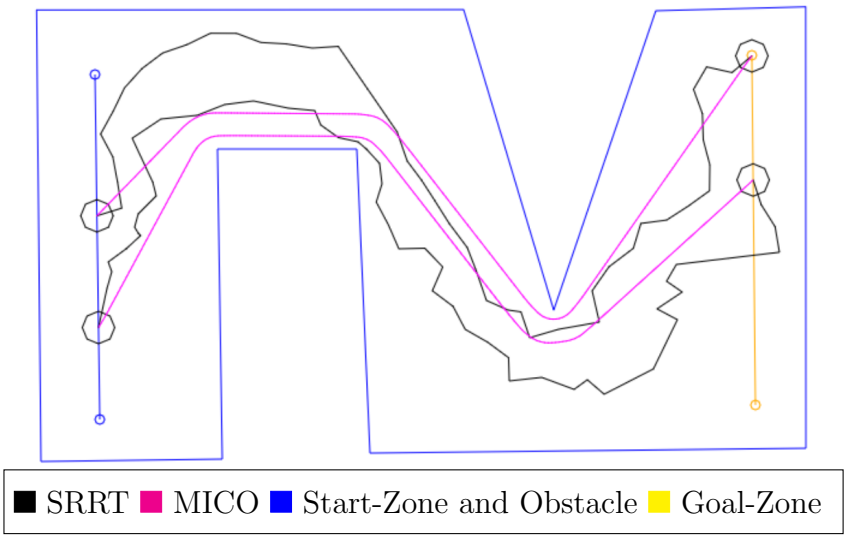


Figure 5.8: *MICO using SRRT input in the "Teeth" environment with* $c_L = 0.50$, *and* $c_{K_{max}} = 0.50$. *The results shows a smoother trajectory near sharp obstacles, with a shorter length compared to initial paths*

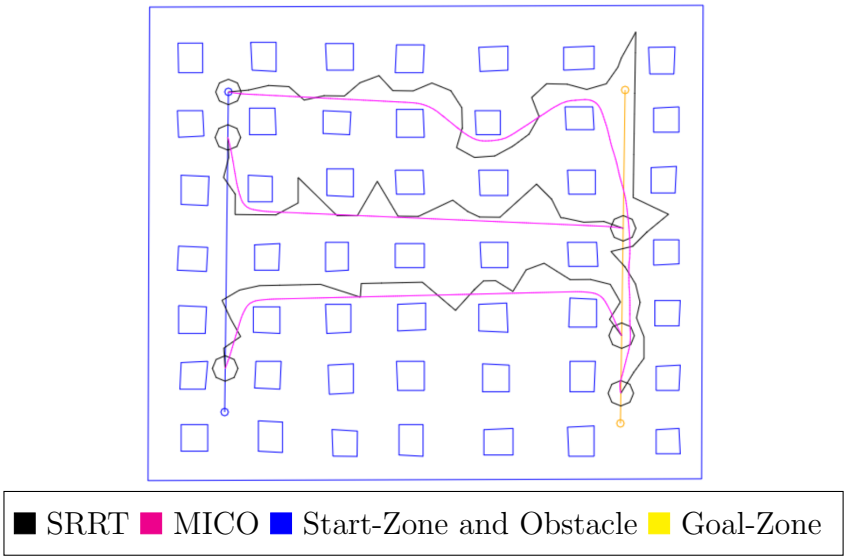


Figure 5.9: *Illustration of the optimized paths of MICO when prioritizing length in the "Regular" environment. Prioritizing length leads to shorter and relatively smoother trajectories, maintaining a safe distance from obstacles (Parameters:* $c_L = 0.70$, $c_{K_{max}} = 0.30$, *and* $c_\Delta = 0.20$*).*

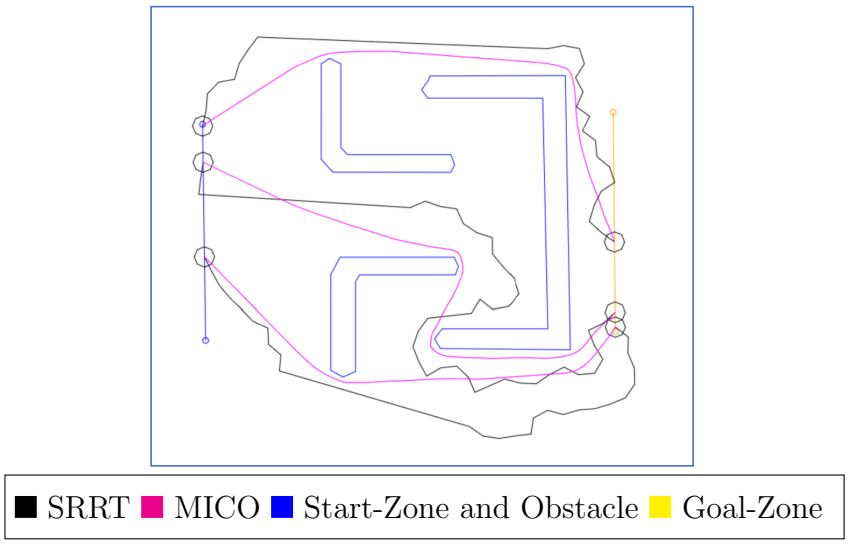


Figure 5.10: *Illustration of the optimized paths of MICO in the "Elbow" environment, with a balanced weight between curvature, length, and clearance. The result indicates a path finely adjusted to minimize length, enhance clearance, and achieve smoothness in vertex degrees.*

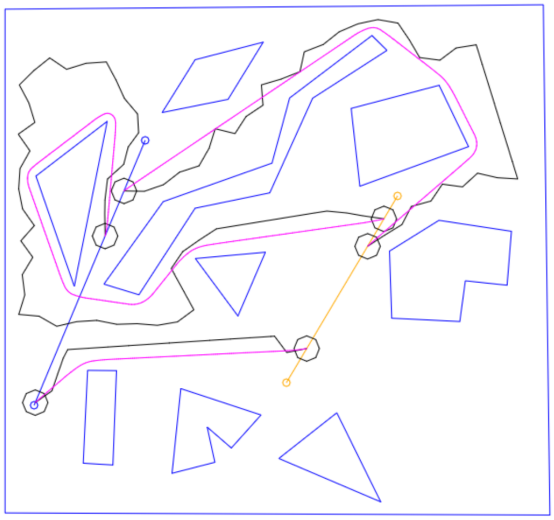

Figure 5.11: *Illustration of the optimized paths of MCO in the "Mixed" environment, with a balanced weight between curvature, length, and clearance. The outcome suggests a path finely tuned for minimized length, reduced clearance, and a pleasing smoothness in vertex degrees.*

| Method | Avg T | Avg L | Max K | Avg K | Min C | Avg C |
|---|---|---|---|---|---|---|
| MBCO | 1.33 | 30.36 | 4.24 | 2.46 | 0.16 | 0.88 |
| MICO | **0.79** | **26.49** | 3.73 | **1.85** | 0.14 | **0.94** |
| MCO | 1.01 | 34.59 | **3.53** | 2.23 | **0.18** | 0.82 |

Table 5.2: Numerical comparison of multi-agent convex optimization techniques in the " Mixed " environment. MICO on average computed the path with the best computational time, path length, and average curvature, and clearance. Notation: T = Time in seconds, K = Curvature, L= Length, and C = Clearance.

| Objective function | Avg T | Avg L | Max K | Avg K | Min C | Avg C |
|---|---|---|---|---|---|---|
| Priority length | **0.23** | **17.03** | 2.31 | 1.74 | 0.17 | 1.40 |
| Priority Curvature | 0.68 | 21.57 | **1.51** | **1.14** | **0.24** | 1.43 |
| Priority Length + Curvature | 0.98 | 19.37 | 2.11 | 1.54 | 0.19 | **1.48** |

Table 5.3: Numerical analysis in the " Teeth" environment. The MICO method is applied with distinct objective functions length, curvature, and considering both length, curvature displays the impact of objective function coefficients on the optimized path quality.

### 5.4.3 Conclusion

In conclusion, the exploration of Multi-agent Interleaving Convex Optimization (MICO) in comparison to methods like MBCO and MCO demonstrates its superior path optimization capability. Our experiments reveal that MICO not only reduces computation time but also exhibits low curvature in diverse environments, including Office, Regular, and simple. The priority function, which is included in MICO, is easy for the user to understand and allows the user the ability to inform the path optimization order with domain-specific knowledge. This method builds upon previously accepted path optimization algorithm in the realm of single agent planning, while extending them in an efficient manner to Multi-agent problems. Our multi-objective function guides the agents' paths to configurations which are optimal in terms of length, curvature, and clearance to obstacles.

The dynamic constraints of robots and agents, specifically in terms of maximum acceleration and velocity, impact the design of optimal paths. Path smoothness, achieved by minimizing bends, becomes crucial for maintaining speed and stability in direction. Although quadratic Bézier curves offer $C^1$ continuity, ensuring $C^0$, and $C^1$ involves solving a linear system for each curve. While quadratic curves simplify curvature integration, Our future research focuses on exploring the applicability of piecewise cubic Bézier curves.

# Chapter 6

# Conclusion

Overall, the presented Deterministic Shortcut-based Smoothing (DSS) method presents a valuable solution when time efficiency is a primary concern. In scenarios where real-time responsiveness or fast trajectory generation is necessary, these methods prove indispensable. By taking advantage of simplified computations and heuristics, shortcut methods facilitate efficient navigation through complex environments. However, it is important to acknowledge that this efficient approach introduce discontinuities in the generated paths, potentially leading to less visually smooth trajectories. On the other hand, using Bézier curves with $C1$ continuity ensures a visually smooth trajectory, which is ideal for scenarios where smoothness matters. The decision for the choice between the presented optimization methods in this thesis depends on the task's priorities: shortcuts can be chosen when time is crucial, and optimization using Bézier curves can be chosen when a smooth and high-quality path is the goal.

The proposed optimization method can be extended to employ Cubic Bézier curves as path segments, rather than quadratic curves. Cubic Bézier curves can provide $C^2$ continuity to the resulting optimized paths. We decided to limit our scope and not use Cubic Bézier curves in our path representation in an effort to reduce the complexity of the convex optimization model. Initial results evaluating the feasibility of cubic or higher-order bezier curves are detailed in Appendix 7.

## 6.1 Future Directions

A promising direction to consider is to address the determination of lanes for multi-agent systems using a Max Flow Algorithm, and then apply the proposed multi-lane optimization methods to improve the results. For example, it is possible to consider a medial axis transform to then employ graph-based max flow algorithms. These variations deal with challenges including assigning capacity at vertex points, handling crossing flows, and using multiple sources and sinks. These directions are promising in terms of achieving sets of lanes of maximum flow.

Another relevant problem deals with Crossing Flows. Crossing Flows happen when lanes addressing different sets of sources and sinks, need to intersect at edges or vertices, possibly causing assigned agents to collide. Even when the edge or vertex in question has enough capacity to accommodate all the agents on it, they may still collide as they need to leave the crossing sections in different directions. Thus, the problem is distinct from the task of assigning or correctly using edge and vertex capacities. Employing the time dimension to prevent agents from occupying the same space at the same time becomes necessary, in order to determine the time of occupancy for every edge and vertex of the path.

In order to address the need for curvature control and $C^2$ continuity in path optimization a clear direction is employ cubic Bézier curves for the path representation. Cubic Bézier curves allow us to independently specify the position and tangent of both endpoints of each segment. In comparison to quadratic Bézier curves, which are only suited to the Corner test we derived for the DSS method, cubic Bézier curves are more versatile and can be used in conjunction with our DSS Disk test.

# Chapter 7

# Appendix

## Definition of Bézier Curves

Let $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n$ be $n+1$ control points in $\mathbb{R}^d$, where $d$ is the dimension of the space in which the curve lies. The Bézier curve of degree $n$ with these control points is defined as

$$\mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{p}_i,$$

where $t \in [0, 1]$ is a parameter that determines the position on the curve.

## Properties of Bézier Curves

Bézier curves have the following properties:

- **Degree:** The degree of a Bézier curve is equal to the number of control points minus one. A curve of degree $n$ requires at least $n+1$ control points.
- **Endpoint interpolation:** The curve passes through the first and last control points, $\mathbf{p}_0$ and $\mathbf{p}_n$. This property is called endpoint interpolation.
- **Affine invariance:** Bézier curves are affine invariant, which means that they are unchanged by affine transformations such as translation, rotation, scaling, and shearing.

- **Convex hull property:** The curve is contained within the convex hull of its control points. This property is called the convex hull property.
- **Local control:** Each segment of the curve between adjacent control points is influenced only by those control points and not by the other control points. This property is called local control.
- **Smoothness:** The curve is smooth, meaning that it has no sharp corners or discontinuities. The degree of smoothness depends on the degree of the curve and the arrangement of the control points.

These properties make Bézier curves useful for a wide range of applications in computer graphics, computer-aided design, and other fields.

## Derivative, Continuity of Bézier Curve

The derivatives of a Bézier curve can be determined geometrically from its control points [15, 88]. The first derivative of a Bézier curve is evaluated as

$$B'(t) = \sum_{i=0}^{n-1} n(\mathbf{p_{i+1}} - \mathbf{p_i}) \binom{n}{i} (1-t)^{n-i} t^i \tag{7.1}$$

Where $n(\mathbf{p_{i+1}} - \mathbf{p_i})$, are control points of $P'(t)$

In order to obtain the higher order derivative of a Bézier curve, the relationship of equation 7.1, can be used iteratively.

Two Bézier curve $B(t)$ and $Q(t)$ are said to be $C^k$ continuous at the endpoint $B(1)$ if if

$$B(1) = Q(0), \quad B'(1) = Q'(0), ..., B^k(1) = Q^k(0) \tag{7.2}$$

Thus, $C^0$ continuity means simply that the two adjacent curves share a common endpoint. $C^1$ continuity means that the two curves not only share the same endpoint, but also that they have the same tangent vector at their shared endpoint, in magnitude as well as in direction. $C^2$ continuity means that two curves have

$C^1$ continuity and, in addition, that they have the same second order parametric derivatives at their shared endpoint, both in magnitude and in direction.

**Lemma 1** If $B(t)$ and $Q(t)$ are at least $C^2$ continuous at $B(1)$, for the path constructed by two Bézier curve segments $B(t)$ and $Q(t)$, then the path has continuous curvature at every point on the curves.

## Distance from line segment of obstacles to quadratic Bézier curve segment

Let $B(t)$ be the equation of quadratic Bézier curve and $\frac{dB}{dt}(t)$ be the derivative of $B(t)$.

$$B(t) = (1-t)^2\mathbf{p}_0 + 2t(1-t)\mathbf{p}_1 + t^2\mathbf{p}_2 \tag{7.3}$$

$$\frac{dB}{dt}(t) = -2(1-t)\mathbf{p}_0 + 2(1-2t)\mathbf{p}_1 + 2t\mathbf{p}_2 \tag{7.4}$$

Let $L(u)$ be equation of line segment (Bézier curve of degree 1) and $\frac{dL}{du}(u)$ be the derivative of $L(u)$.

$$L(u) = (1-u)\mathbf{l}_0 + u\mathbf{l}_1 \tag{7.5}$$

$$\frac{dL}{du}(u) = -\mathbf{l}_0 + \mathbf{l}_1 \tag{7.6}$$

Since tangent vectors $\frac{dB}{dt}$ and $\frac{dL}{du}$ are co-linear, we have the following equation where $\alpha$ is a scalar:

$$\frac{dB}{dt} = \alpha\frac{dL}{du} \tag{7.7}$$

In order to find the shortest distance between quadratic Bézier curve $B(t)$ and line segment $L(t)$, we need to find the following:

1. The closest point $B^*$ on the Bézier curve to the line segment. To find this, we will find value of $t$ that satisfies at $B(t)$.

2. The closest point $l^*$ on the line segment $L$ to the Bézier curve.

Using the formulas above, we can find $B^*$ and $L^*$. If $B^*$ falls outside of the Bézier curve segment $B(t)$, then the distance from $B^*$ to $L^*$ is not valid. We should instead check the endpoints of Bézier curve $B(t)$. We should be careful to find the distance between $B_0$ and line segment $L(t)$ and between $B(1)$ and line segment $L$.

We obtain shortest distance from point $L^*$ on the line segment to the Bézier curve $B(t)$ as following:
if $0 <= t^* <= 1$ and $0 <= u^* <= 1$:
return $(p^*, l^*)$
else:
return $min(dist(L, p_0), dist(L, p_1))$

Note that for the last line which returns the minimum distance, we call a subroutine for calculating the distance between a point and a line segment.

# Curvature of a quadratic Bézier Curve

The curvature for a parameterized curve $B(t) = ((x(t), y(t))$ is given by:

$$K(t) = \frac{|B'(t), B''(t)|}{\|B'(t)\|^3}$$

Where the numerator is the determinant of the matrix created by concatenating $B'(t)$ and $B''(t)$.

A quadratic Bézier curve is defined by the points $\mathbf{p}_0$, $\mathbf{p}_1$, and $\mathbf{p}_2$ is parameterized by

$$B(t) = (1-t)^2\mathbf{p}_0 + 2(1-t)t\mathbf{p}_1 + t^2\mathbf{p}_2 \tag{7.8}$$

with derivatives

$$B'(t) = 2(1-t)(\mathbf{p}_1 - \mathbf{p}_0) + 2t(\mathbf{p}_2 - \mathbf{p}_1) \tag{7.9}$$

$$B''(t) = 2(\mathbf{p}_2 - 2\mathbf{p}_1 + \mathbf{p}_0) \tag{7.10}$$

By using the bi-linearity of the determinant operator and the fact that $|x, x| \equiv 0$, the numerator given by $n(t) = |B'(t), B''(t)|$ can be obtained by plugging

these values into the expression for the curvature. To simplify the calculation, we introduce the variables $s(t)$ and $w(t)$:

$$\begin{aligned} s(t) &= 4(1-t)\,|\mathbf{p}_1 - \mathbf{p}_0, \mathbf{p}_2 - \mathbf{p}_1| \\ &\quad + 4t\,|\mathbf{p}_2 - \mathbf{p}_1, \mathbf{p}_0 - \mathbf{p}_1| \\ s(t) &= 4\,|P_1 - P_0, P_2 - P_1| \end{aligned} \tag{7.11}$$

The denominator of 7 is given by

$$w(t) = \|B(t)\|^3$$

where $|B(t)|^2$ can be expanded as following:

$$\begin{aligned} w(t) &= 4(1-t)^2\,|\mathbf{p}_1 - \mathbf{p}_0|^2 \\ &\quad + 8t(1-t)(\mathbf{p}_1 - \mathbf{p}_0)\cdot(\mathbf{p}_2 - \mathbf{p}_1) \\ &\quad + 4t^2\,|\mathbf{p}_2 - \mathbf{p}_1|^2 \end{aligned} \tag{7.12}$$

Either the highest degree of curvature is located at (i) the peak of the function $K(t)$ or (ii) at one of the endpoints of the curve if the peak is beyond the range of (0,1). The maximum of the function $K(t)$ is associated with $K'(t) = 0$, meaning that the slope of the curvature is zero at that point.

The equation $K'(t) = \frac{s'(t)w(t) - s(t)w'(t)}{w(t)^2}$ shows that to find the zeros of $K'(t)$, one must find the zeros of $w'(t)$ since $w(t)$ is constant. Finding the zeros of $w'(t)$ further simplifies to finding the zeros of $\|B'(t)\|^2$. This can be computed using the formula $\frac{d}{dt}\|B(t)\|^2 = 8(\mathbf{p}_1 - \mathbf{p}_0)\cdot(\mathbf{p}_0 - 2\mathbf{p}_1 + \mathbf{p}_2) + 8t||\mathbf{p}_0 - 2\mathbf{p}_1 + \mathbf{p}_2||$, which yields the optimal parameter value $t^* = \frac{(\mathbf{p}_1 - \mathbf{p}_0)\cdot(\mathbf{p}_0 - 2\mathbf{p}_1 + \mathbf{p}_2)}{||\mathbf{p}_0 - 2\mathbf{p}_1 + \mathbf{p}_2||}$. Plugging this value into the expression and performing some algebraic manipulation gives the following formula for the maximum curvature of a quadratic Bézier Curve:

$$K(t^*) = \frac{||\mathbf{p}_2 - 2\mathbf{p}_1 + \mathbf{p}_0||}{2|\mathbf{p}_1 - \mathbf{p}_0, \mathbf{p}_2 - \mathbf{p}_1|} \tag{7.13}$$

# Approximation of Arc Length of Quadratic Bézier Curves

To compute the arc length of a curve, we need to integrate the magnitude of its derivative over the parameter interval:

$$L = \int_0^1 \left\| \frac{dB}{dt} \right\| dt \tag{7.14}$$

We can derive the following expression by expanding equation 7.14:

$$\left\| \frac{dB}{dt} \right\| = \sqrt{e_1^2 + e_2^2} \tag{7.15}$$

where $e_1$, and $e_2$ are defined as below:

$$e_1 = (2(1-t)(x_1 - x_0) + 2t(x_2 - x_1))^2$$
$$e_2 = (2(1-t)(y_1 - y_0) + 2t(y_2 - y_1))^2$$

Substituting the second degree polynomial form of the expression inside the square root of 7.15 into 7.14 simplifies the integral and makes it easier to solve. To express the equation as a second degree polynomial, we take a factor from $t$ and $t^2$ while expanding 7.15.

$$\begin{aligned}
e_1^2 + e_2^2 &= [2(1-t)(x_1 - x_0) + 2t(x_2 - x_1)]^2 \\
&\quad + [2(1-t)(y_1 - y_0) + 2t(y_2 - y_1)]^2 \\
&= t^2[4x_2^2 + 16x_1^2 + 4x_0^2 - 16x_2x_1 + 8x_2x_0 \\
&\quad - 16x_1x_0 + 4y_2^2 + 16y_1^2 + 4y_0^2 - 16y_2y_1 \\
&\quad + 8y_2y_0 - 16y_1y_0] + t[16x_1^2 - 8x_0^2 + 8x_2x_1 \\
&\quad - 8x_2x_0 + 24x_1x_0 - 16y_1^2 - 8y_0^2 + 8y_2y_1 \\
&\quad - 8y_2y_0 + 24y_1y_0] + [-8x_1x_0 + 4x_0^2 + 4x_1^2 \\
&\quad - 8y_1y_0 + 4y_0^2 + 4y_1^2]
\end{aligned} \tag{7.16}$$

To simplify the calculation of arc length, we establish the constants $a$, $b$, and $c$ in the following manner:

$$\begin{aligned} a &= 4x_2^2 + 16x_1^2 + 4x_0^2 - 16x_2x_1 + 8x_2x_0 - 16x_1x_0 \\ &\quad + 4y_2^2 + 16y_1^2 + 4y_0^2 - 16y_2y_1 + 8y_2y_0 - 16y_1y_0 \\ b &= -16x_1^2 - 8x_0^2 + 8x_2x_1 - 8x_2x_0 + 24x_1x_0 \\ &\quad - 16y_1^2 - 8y_0^2 + 8y_2y_1 - 8y_2y_0 + 24y_1y_0 \\ c &= -8x_1x_0 + 4x_0^2 + 4x_1^2 - 8y_1y_0 + 4y_0^2 + 4y_1^2 \end{aligned} \tag{7.17}$$

The equation in 7.14 can therefore be written as:

$$L = \int_0^1 \left\| \frac{dB}{dt} \right\| = \int_0^1 \sqrt{at^2 + bt + c}\, dt \tag{7.18}$$

To solve 7.18, we modify the expression $ax^2 + bx + c$ in the following way:

$$\begin{aligned} ax^2 + bx + c &= a\left(x^2 + \frac{b}{a}x + \frac{c}{a} + \frac{b}{(2a)^2} - \frac{b}{(2a)^2}\right) \\ &= a\left((x + \frac{b}{2a})^2 + (\frac{4ac - b^2}{4a^2})\right) \end{aligned} \tag{7.19}$$

After substituting 7.19 into 7.18, the following computations are carried out:

$$\begin{aligned} L &= \int_0^1 \sqrt{at^2 + bt + c}\, dx \\ &= \int_0^1 \sqrt{a\left((x + \frac{b}{2a})^2 + (\frac{4ac - b^2}{4a^2})\right)}\, dx \\ &= \sqrt{a}\int_0^1 \sqrt{(\frac{4ac - b^2}{4a^2})(\frac{(x + \frac{b}{2a})^2}{\frac{4ac - b^2}{4a^2}}) + 1}\, dx \\ &= \sqrt{a}\sqrt{(\frac{4ac - b^2}{4a^2})}\int_0^1 \sqrt{(\frac{(x + \frac{b}{2a})^2(4a^2)}{4ac - b^2}) + 1}\, dx \\ &= \sqrt{(\frac{4ac - b^2}{4a})}\int_0^1 \sqrt{(\frac{(2ax + b)^2}{4ac - b^2}) + 1}\, dx \end{aligned}$$

If we define $A = \sqrt{\frac{4ac-b^2}{4a}}$, then the equation mentioned above can be written in the following manner:

$$L = A \int_0^1 \sqrt{(\frac{(2ax+b)^2}{4ac-b^2}) + 1}\, dx \tag{7.20}$$

Assuming the expression below, we can solve the integral using a change of variables:

$$\tan(x) = \sqrt{(\frac{(2ax+b)^2}{4ac-b^2})} = \frac{(2ax+b)}{\sqrt{4ac-b^2}} \tag{7.21}$$

Taking the derivative of each side results in

$$\sec^2(t)dt = \frac{2a}{\sqrt{4ac-b^2}}\ dx \tag{7.22}$$

The equation above is equivalent to the one presented below:

$$\frac{\sqrt{4ac-b^2}}{2a} \sec^2(t)\ dt = dx \tag{7.23}$$

By substituting equation 7.23 into equation 7.20 and taking into account that $\sec(t) = \sqrt{\tan^2(t)+1}$, the resulting expression is as follows:

$$\begin{aligned} L &= A \int_0^1 \sqrt{\tan^2(t)+1} \times \frac{\sqrt{4ac-b^2}}{2a} \sec^2(t)\, dt \\ &= A \frac{\sqrt{4ac-b^2}}{2a} \int_0^1 \sec^3(t)\, dt \end{aligned} \tag{7.24}$$

We know that

$$\int_0^1 \sec^3(t)\, dt = \frac{\sec(t)\tan(t)}{2} + \ln \frac{\sec(t)+\tan(t)}{2} \tag{7.25}$$

Substituting 7.25 in 7.24 turns out the following:

$$L = A\frac{\sqrt{4ac-b^2}}{2a}\int_0^1 \sec^3(t)\,dt$$
$$= \frac{4ac-b^2}{4a^{\frac{3}{2}}}\left(\frac{\sec(t)\tan(t)}{2} + ln\left|\frac{\sec(t)+\tan(t)}{2}\right)\right| \tag{7.26}$$

Using $\sec(t) = \sqrt{\tan^2(t)+1}$, the change of variable we did and some algebra 7.26 follows the following equation:

$$L = \frac{4ac-b^2}{8a^{\frac{3}{2}}}\left(\frac{2ax+b}{\sqrt{4ac-b^2}} \times \sqrt{\frac{2ax+b}{4ac-b^2}+1}\right.$$
$$\left. \times\, ln\left|\sqrt{\frac{2ax+b}{4ac-b^2}+1}\right| + \frac{2ax+b}{\sqrt{4ac-b^2}}\right. \tag{7.27}$$

The equation to determine the arc length of a quadratic Bézier curve can be obtained by using the following expression:

$$\text{Arc Length} = I(x=1) - I(x=0) \tag{7.28}$$

where $B$ is defined by:

$$I = w_1\left(w_2 \cdot w_3 + \log|w_3 + w_2|\right) \tag{7.29}$$

To evaluate the constants $a$, $b$, $c$, $n_1$, $n_2$, $w_1$, $w_2$, and $w_3$, we use the following equations:

$$n_1 = 2ax - b$$
$$n_2 = 4ac - b^2$$
$$w_1 = \frac{n_2}{8a^{\frac{3}{2}}}$$
$$w_2 = \frac{n_1}{\sqrt{n_2}}$$
$$w_3 = \sqrt{\frac{n_1^2}{n_2^2}+1}$$

The values for $a$, $b$, and $c$ are established in equation 7.17.

## Cubic Bézier approach

In order to address the need for curvature control and $C^2$ continuity in path optimization we propose the use of cubic Bézier curves. Cubic Bézier curve shortcuts allow us to independently specify the position and tangent of both endpoints of the shortcut. In comparison to quadratic Bézier curves, which are only suited to the Corner test we derived, cubic Bézier curves are more versatile and can be used in conjunction with our Disk test. Figure 7.4 demonstrates a shortcut which is not possible to create using a single quadratic Bézier curve while satisfying $C^2$ continuity on both endpoints. Thus, cubic Bézier curves have the potential to make longer shortcut. We outline the procedure for choosing cubic Bézier curves, which follows from Disk test from our preliminary work:

1. As previously, we find the vertex with most free space which we label $\mathbf{v}^*$. We denote the distance from $\mathbf{v}^*$ to the nearest obstacle as $d^*$.

2. We find the intersections between the path and the circle $C$ of radius $d^*$ centered at $\mathbf{v}^*$, and label them $\mathbf{P_0}$ and $\mathbf{P_3}$, where $\mathbf{P_3}$ occurs later in the path. We call these two control points the position points.

3. We determine the tangent lines for both $\mathbf{P_0}$ and $\mathbf{P_3}$. We extend the tangent line from $\mathbf{P_0}$ into the circle $C$ by a fixed distance $d_f$, and call the end point of this tangent line $\mathbf{P}'$. We also find the next vertex along the path after $\mathbf{P_0}$, which we call $\mathbf{v}'$. If $d_f$ is greater than the radius of the circle, we assign the tangent control point $\mathbf{P_1}$ to be $\mathbf{v}'$. Otherwise, we let $\mathbf{P_1} = \mathbf{P}'$. We perform the same process from $\mathbf{P_3}$ to determine $\mathbf{P_2}$. We call $\mathbf{P_1}$ and $\mathbf{P_2}$ the tangent points of $\mathbf{P_0}$ and $\mathbf{P_3}$, respectively.

In the following figures, we illustrate several of the advantages of the cubic Bézier approach.

In 7.2, the circle around $\mathbf{v}^*$ is small, so this cubic method behaves like a quadratic Bézier curve because $\mathbf{P_1}$ and $\mathbf{P_2}$ are the same point. This minimizes curvature around sharp corners. Some arrangements of control points result in "shortcuts" which are longer than the original path, is shown in part (b) of 7.2.

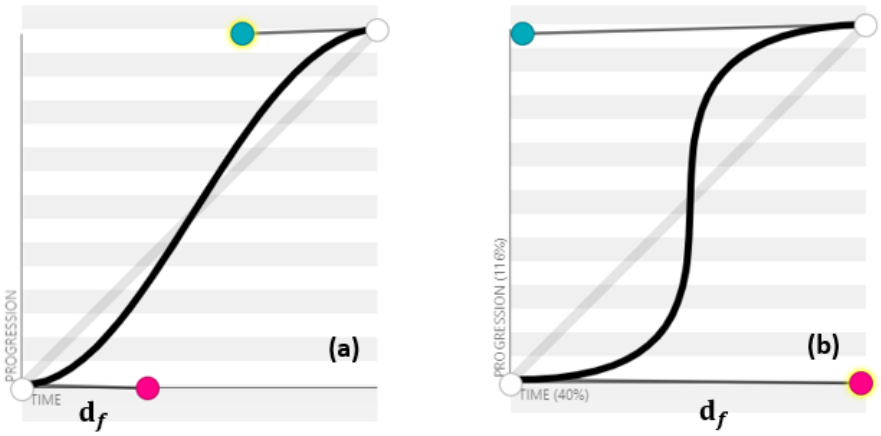


Figure 7.1: *Sub-figure (a) shows a shortcut with a small distance between endpoints and tangent control points, $d_f$, resulting in a small radius of curvature near the end points. Sub-figure (b) shows the effect of choosing a large $d_f$, as the shortcut has larger radius of curvature.*

In these cases, we choose v* or another vertex on the path as the tangent control points. In figure 7.3, the circle is much larger than $d_f$, which makes the resulting shortcut appear close to flat. This minimizes the length of the shortcut while maintaining $C^2$ continuity. As in previous algorithms, when there are not two points of intersections between the circle and the path, we can choose the relevant path endpoint as the shortcut end point $\mathbf{P_0}$ or $\mathbf{P_3}$. Figure 7.4 demonstrates one such case while also showing the flexibility of cubic Bézier curves in creating shortcuts regardless of tangent lines. $d_f$ can be increased to favor smaller curvature where possible in the path or decrease to favor minimizing length. Figure 7.1 demonstrates the effect of changing $d_f$. When the direction of the tangent at both endpoints of the shortcut is fixed, the curvature of the shortcut can still be controlled by choosing how far away the tangent control points are from the endpoints.

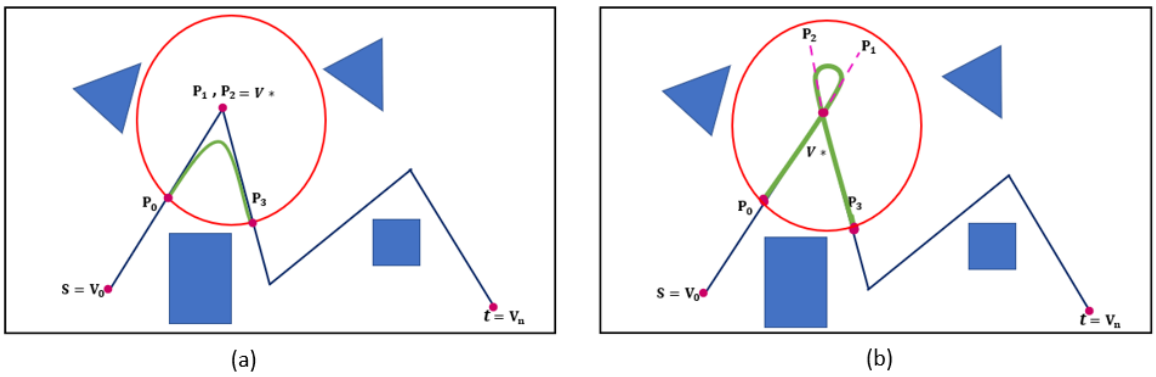


Figure 7.2: *Some arrangements of control points result in "shortcuts" which are longer than the original path, as shown in (b). In these cases, we choose $v^*$ or another vertex on the path as the tangent control points as shown in (a).*

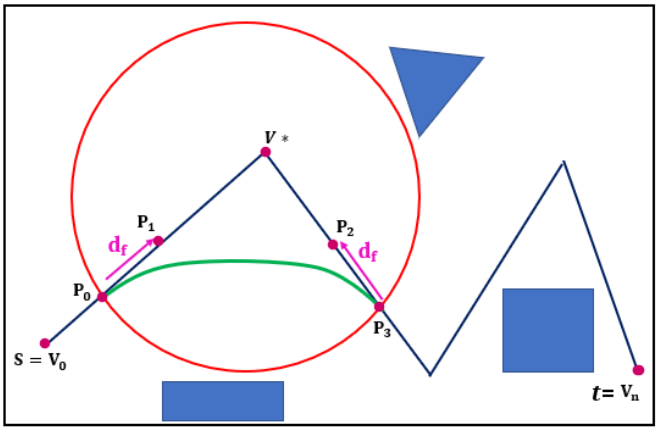


Figure 7.3: *We identify a fixed distance in order to produce a shortcut like a straight line which make the length of the short smoother. When using a fixed distance between $P_0$ and $P_1$, the generated shortcut is close to a straight line shortcut, which provides the best length shortening while maintaining $C^2$ continuity.*

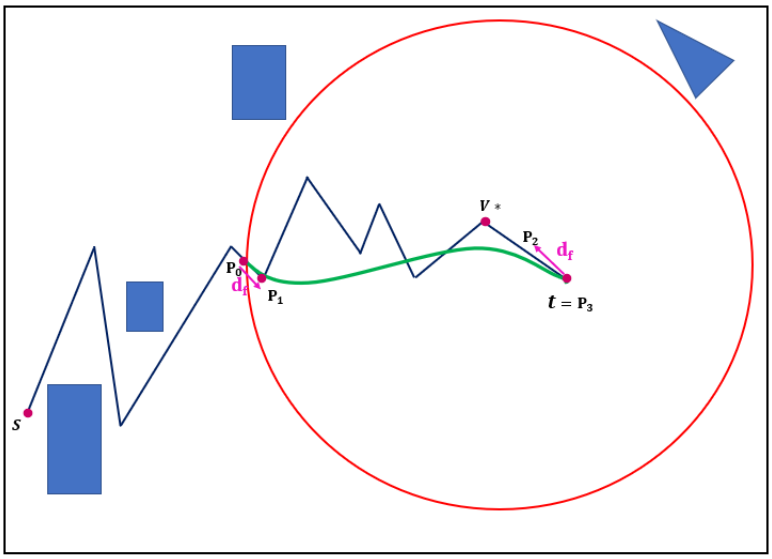


Figure 7.4: *Cubic Bezier curves create shortcuts between path segments with any tangent. The tangent lines from the end points of the shortcut are taken to point into the circle.*

# Bibliography


[1] Sheelan Waad Adwaan et al. Proposed modified directed RRT algorithm of the basic RRT. *Al-Mansour Journal*, 33, 2020.

[2] Shubhani Aggarwal and Neeraj Kumar. Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges. *Computer Communications*, 149:270–299, 2020.

[3] Ali-Akbar Agha-Mohammadi, Suman Chakravorty, and Nancy M Amato. Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *The International Journal of Robotics Research*, 33(2):268–304, 2014.

[4] Ebtehal Turki Saho Alotaibi and Hisham Al-Rawi. Multi-robot path-planning problem for a heavy traffic control application: A survey. *International Journal of Advanced Computer Science and Applications*, 7(6), 2016.

[5] Federico Augugliaro, Angela P Schoellig, and Raffaello D'Andrea. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*, pages 1917–1922. IEEE, 2012.

[6] Gustavo SC Avellar, Guilherme AS Pereira, Luciano CA Pimenta, and Paulo Iscold. Multi-uav routing for area coverage and remote sensing with minimum time. *Sensors*, 15(11):27783–27803, 2015.

[7] Hyansu Bae, Gidong Kim, Jonguk Kim, Dianwei Qian, and Sukgyu Lee. Multi-robot path planning method using reinforcement learning. *Applied sciences*, 9(15):3057, 2019.

[8] Mylène Campana, Florent Lamiraux, and Jean-Paul Laumond. A simple path optimization method for motion planning. working paper or preprint, September 2015.

[9] Runqi Chai, Al Savvaris, Antonios Tsourdos, Senchun Chai, Runqi Chai, Al Savvaris, Antonios Tsourdos, and Senchun Chai. Overview of trajectory

optimization techniques. *Design of Trajectory Optimization Approach for Space Maneuver Vehicle Skip Entry Problems*, pages 7–25, 2020.

[10] Runqi Chai, Al Savvaris, Antonios Tsourdos, Senchun Chai, and Yuanqing Xia. Improved gradient-based algorithm for solving aeroassisted vehicle trajectory optimization problems. *Journal of Guidance, Control, and Dynamics*, 40(8):2093–2101, 2017.

[11] Jung-Woo Chang, Yi-King Choi, Myung-Soo Kim, and Wenping Wang. Computation of the minimum distance between two bézier curves/surfaces. *Computers & Graphics*, 35(3):677–684, 2011.

[12] Gang Chen, Ning Luo, Dan Liu, Zhihui Zhao, and Changchun Liang. Path planning for manipulators based on an improved probabilistic roadmap method. *Robotics and Computer-Integrated Manufacturing*, 72:102196, 2021.

[13] Hailong Chen, Qiang Wang, Meng Yu, Jingjing Cao, and Jingtao Sun. Path planning for multi-robot systems in intelligent warehouse. In *Internet and Distributed Computing Systems: 11th International Conference, IDCS 2018, Tokyo, Japan, October 11–13, 2018, Proceedings 11*, pages 148–159. Springer, 2018.

[14] Sung-Won Cho, Jin-Hyoung Park, Hyun-Ji Park, and Seongmin Kim. Multi-uav coverage path planning based on hexagonal grid decomposition in maritime search and rescue. *Mathematics*, 10(1):83, 2021.

[15] Ji-wung Choi, R Curry, and Gabriel Hugh Elkaim. Smooth path generation based on bézier curves for autonomous vehicles. In *WCECS (World Congress on Engineering and Computer Science 2009 Vol II)*, 2009.

[16] Ji-wung Choi, Renwick Curry, and Gabriel Elkaim. Path planning based on bézier curve for autonomous ground vehicles. In *Advances in Electrical and Electronics Engineering-IAENG Special Edition of the World Congress on Engineering and Computer Science 2008*, pages 158–166. IEEE, 2008.

[17] Ji-Wung Choi, Renwick Curry, and Gabriel Elkaim. Piecewise bezier curves path planning with continuous curvature constraint for autonomous driving. In *Machine learning and systems engineering*, pages 31–45. Springer, 2010.

[18] Ji-wung Choi, Renwick E Curry, and Gabriel H Elkaim. Minimizing the maximum curvature of quadratic bézier curves with a tetragonal concave polygonal boundary constraint. *Computer-Aided Design*, 44(4):311–319, 2012.

[19] Ji-wung Choi and Gabriel Hugh Elkaim. Bézier curves for trajectory guidance. In *World Congress on Engineering and Computer Science, WCECS*, pages 22–24. Citeseer, 2008.

[20] Younsung Choi, Donghyung Kim, Soonwoong Hwang, Hyeonguk Kim, Namwun Kim, and Changsoo Han. Dual-arm robot motion planning for collision avoidance using B-spline curve. *International Journal of Precision Engineering and Manufacturing*, 18(6):835–843, 2017.

[21] Reinis Cimurs, Jaepyung Hwang, and Il Hong Suh. Bezier curve-based smoothing for path planner with curvature constraint. In *2017 First IEEE International Conference on Robotic Computing (IRC)*, pages 241–248. IEEE, 2017.

[22] Juan Dai and Yuanqing Xia. Mars atmospheric entry guidance for reference trajectory tracking. *Aerospace science and technology*, 45:335–345, 2015.

[23] SCMS De Sirisuriya, TGI Fernando, and MKA Ariyaratne. Algorithms for path optimizations: a short survey. *Computing*, 105(2):293–319, 2023.

[24] Robin Deits and Russ Tedrake. Efficient mixed-integer planning for uavs in cluttered environments. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 42–49. IEEE, 2015.

[25] JFC Dekkers. Application of bezier curves in computer-aided design. *Delft Institute of Applied Mathematics*, 2010.

[26] Moritz Diehl, Hans Georg Bock, Holger Diedam, and P-B Wieber. Fast direct multiple shooting algorithms for optimal robot control. *Fast motions in biomechanics and robotics: optimization and feedback control*, pages 65–93, 2006.

[27] 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.

[28] SA Eshtehardian and S Khodaygan. A continuous rrt*-based path planning method for non-holonomic mobile robots using b-spline curves. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–10, 2022.

[29] Fariba Fahroo and I Michael Ross. Direct trajectory optimization by a chebyshev pseudospectral method. *Journal of Guidance, Control, and Dynamics*, 25(1):160–166, 2002.

[30] Renato Farias and Marcelo Kallmann. Improved shortest path maps with gpu shaders. *arXiv preprint arXiv:1805.08500*, 2018.

[31] Liping Fu, Dihua Sun, and Laurence R Rilett. Heuristic shortest path algorithms for transportation applications: State of the art. *Computers & Operations Research*, 33(11):3324–3343, 2006.

[32] Zhanna Gabbassova, Davoud Sedighizadeh, Alireza Sheikhi Fini, and Mostafa Seddighizadeh. Multiple robot motion planning considering shortest and safest trajectory. *Electromechanical Energy Conversion Systems*, 1(3):1–6, 2021.

[33] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed rrt: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ international conference on intelligent robots and systems*, pages 2997–3004. IEEE, 2014.

[34] Roland Geraerts and Mark H Overmars. Clearance based path optimization for motion planning. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2386–2392. IEEE, 2004.

[35] Roland Geraerts and Mark H Overmars. Creating high-quality paths for motion planning. *The international journal of robotics research*, 26(8):845–863, 2007.

[36] Andrew V Goldberg. Point-to-point shortest path algorithms with preprocessing. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 88–102. Springer, 2007.

[37] Reda Guernane and Mahmoud Belhocine. A smoothing strategy for PRM paths application to six-axes MOTOMAN SV3X manipulator. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4155–4160. IEEE, 2005.

[38] Faiza Gul, Imran Mir, Laith Abualigah, Putra Sumari, and Agostino Forestiero. A consolidated review of path planning and optimization techniques: Technical perspectives and future directions. *Electronics*, 10(18):2250, 2021.

[39] Kris Hauser and Victor Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *2010 IEEE International Conference on Robotics and Automation*, pages 2493–2498. IEEE, 2010.

[40] Eric Heiden, Luigi Palmieri, Sven Koenig, Kai O Arras, and Gaurav S Sukhatme. Gradient-informed path smoothing for wheeled mobile robots. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1710–1717. IEEE, 2018.

[41] Matthias Heinkenschloss and Denis Ridzal. A matrix-free trust-region sqp method for equality constrained optimization. *SIAM Journal on Optimization*, 24(3):1507–1541, 2014.

[42] Martin Holzer, Frank Schulz, Dorothea Wagner, and Thomas Willhalm. Combining speed-up techniques for shortest-path computations. *Journal of Experimental Algorithmics (JEA)*, 10:2–5, 2005.

[43] David Hsu, J-C Latcombe, and Stephen Sorkin. Placing a robot manipulator amid obstacles for optimized execution. In *Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning (ISATP'99)(Cat. no. 99TH8470)*, pages 280–285. IEEE, 1999.

[44] Sheng-Kai Huang, Wen-June Wang, and Chung-Hsun Sun. A path planning strategy for multi-robot moving with path-priority order based on a generalized voronoi diagram. *Applied Sciences*, 11(20):9650, 2021.

[45] Shuai Huang, Dingkang Yang, Chuyi Zhong, Shichao Yan, and Lihua Zhang. An improved ant colony optimization algorithm for multi-agent path planning. In *2021 International Conference on Networking Systems of AI (INSAI)*, pages 95–100. IEEE, 2021.

[46] Jingchao Jiang and Yongsheng Ma. Path planning strategies to optimize accuracy, quality, build time and material use in additive manufacturing: a review. *Micromachines*, 11(7):633, 2020.

[47] Yuan Jin, Jianke Du, Zhiyong Ma, Anbang Liu, and Yong He. An optimization approach for path planning of high-quality and uniform additive manufacturing. *The International Journal of Advanced Manufacturing Technology*, 92:651–662, 2017.

[48] Timothy R Jorris and Richard G Cobb. Three-dimensional trajectory optimization satisfying waypoint and no-fly zone constraints. *Journal of Guidance, Control, and Dynamics*, 32(2):551–572, 2009.

[49] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*, pages 4569–4574. IEEE, 2011.

[50] Marcelo Kallmann, Amaury Aubel, Tolga Abaci, and Daniel Thalmann. Planning collision-free reaching motions for interactive object manipulation and grasping. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, New York, NY, USA, 2008. Association for Computing Machinery.

[51] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[52] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[53] Lydia E Kavraki, Mihail N Kolountzakis, and J-C Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and automation*, 14(1):166–171, 1998.

[54] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[55] Taro Kawasaki, Pradeep Kumar Jayaraman, Kentaro Shida, Jianmin Zheng, and Takashi Maekawa. An image processing approach to feature-preserving b-spline surface fairing. *Computer-Aided Design*, 99:1–10, 2018.

[56] Maryam Khazaei Pool, Matthew Morozov, and Marcelo Kallmann. Optimizing curvature and clearance of piecewise bézier paths. *IEEE ICCMA 2023*, NA:NA, 2023.

[57] Calvin Kielas-Jensen, Venanzio Cichella, Thomas Berry, Isaac Kaminer, Claire Walton, and Antonio Pascoal. Bernstein polynomial-based method for solving optimal trajectory generation problems. *Sensors*, 22(5):1869, 2022.

[58] Dong-Hyung Kim, Youn-Sung Choi, Sang-Ho Kim, Jing Wu, Chao Yuan, Lu-Ping Luo, Ji Yeong Lee, and Chang-Soo Han. Adaptive rapidly-exploring random tree for efficient path planning of high-degree-of-freedom articulated robots. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 229(18):3361–3367, 2015.

[59] Boris Lau, Christoph Sprunk, and Wolfram Burgard. Kinodynamic motion planning for mobile robots using splines. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2427–2433. IEEE, 2009.

[60] Julien Laurent-Varin, J Frederic Bonnans, Nicolas Bérend, Mounir Haddou, and Christophe Talbot. Interior-point approach to trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 30(5):1228–1238, 2007.

[61] Steven M. LaValle. Rapidly-exploring random trees : a new tool for path planning. *The annual research report*, 1998.

[62] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[63] Juan Li and Chengyu Yang. AUV Path Planning Based on Improved RRT and Bézier Curve Optimization. In *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 1359–1364, Beijing, China, October 2020. IEEE.

[64] Likun Li, Yinsheng Fu, Kun Yu, Ahmed M Alwakeel, and Lubna A Alharbi. Optimal trajectory uav path design based on bezier curves with multi-hop cluster selection in wireless networks. *Wireless Networks*, pages 1–12, 2022.

[65] Xin Li and Thomas W. Sederberg. S-splines: A simple surface solution for iga and cad. *Computer Methods in Applied Mechanics and Engineering*, 350:664–678, 2019.

[66] Libretexts. Sampling-based path planning. *Libretexts*, n.d. In Introduction to Autonomous Robots (Correll). Retrieved November 12, 2023, from `https://eng.libretexts.org/Bookshelves/Mechanical_Engineering/Introduction_to_Autonomous_Robots_(Correll)/04%3A_Path_Planning/4.03%3A_Sampling-based_Path_Planning`.

[67] Xinfu Liu, Ping Lu, and Binfeng Pan. Survey of convex optimization for aerospace applications. *Astrodynamics*, 1:23–40, 2017.

[68] Xinfu Liu, Zuojun Shen, and Ping Lu. Entry trajectory optimization by second-order cone programming. *Journal of Guidance, Control, and Dynamics*, 39(2):227–241, 2016.

[69] Henrik Lurz, Tobias Recker, and Annika Raatz. Spline-based path planning and reconfiguration for rigid multi-robot formations. *Procedia CIRP*, 106:174–179, 2022.

[70] Xiaolu Ma, Rui Gong, Yibo Tan, Hong Mei, Chengcheng Li, et al. Path planning of mobile robot based on improved prm based on cubic spline. *Wireless Communications and Mobile Computing*, 2022, 2022.

[71] Amgad Madkour, Walid G Aref, Faizan Ur Rehman, Mohamed Abdur Rahman, and Saleh Basalamah. A survey of shortest-path algorithms. *arXiv preprint arXiv:1705.02044*, 2017.

[72] Sidra Maqsood, Muhammad Abbas, Kenjiro T Miura, Abdul Majeed, and Azhar Iqbal. Geometric modeling and applications of generalized blended trigonometric bézier curves with shape parameters. *Advances in difference equations*, 2020(1):1–18, 2020.

[73] Ivo Marinić-Kragić, Stipe Perišić, Damir Vučina, and Milan Ćurković. Superimposed rbf and b-spline parametric surface for reverse engineering applications. *Integrated Computer-Aided Engineering*, 27(1):17–35, 2020.

[74] Reza Mashayekhi, Mohd Yamani Idna Idris, Mohammad Hossein Anisi, Ismail Ahmedy, and Ihsan Ali. Informed rrt*-connect: An asymptotically optimal single-query path planning method. *IEEE Access*, 8:19842–19852, 2020.

[75] Imran Mir, Faiza Gul, Suleman Mir, Mansoor Ahmed Khan, Nasir Saeed, Laith Abualigah, Belal Abuhaija, and Amir H Gandomi. A survey of trajectory planning techniques for autonomous systems. *Electronics*, 11(18):2801, 2022.

[76] A A Neto, D G Macharet, and M F M Campos. Feasible RRT-based path planning using seventh order bézier curves. In *IEEE Proceedings*. IEEE, 2010.

[77] Ngoc Thinh Nguyen, Pranav Tej Gangavarapu, Niklas Fin Kompe, Georg Schildbach, and Floris Ernst. Navigation with polytopes: A toolbox for optimal path planning with polytope maps and b-spline curves. *Sensors*, 23(7):3532, 2023.

[78] Iram Noreen. Collision free smooth path for mobile robots in cluttered environment using an economical clamped cubic b-spline. *Symmetry*, 12(9):1567, 2020.

[79] Radhakant Padhi and SN Balakrishnan. Optimal dynamic inversion control design for a class of nonlinear distributed parameter systems with continuous and discrete actuators. *IET control theory & applications*, 1(6):1662–1671, 2007.

[80] Jia Pan, Liangjun Zhang, Dinesh Manocha, and UC Hill. Collision-free and curvature-continuous path smoothing in cluttered environments. *Robotics: Science and Systems VII*, 17:233, 2012.

[81] Probabilistic Roadmap Planning. On the probabilistic foundations of probabilistic roadmap planning. In *Robotics Research: Results of the 12th International Symposium ISRR*, volume 28, page 83. Springer Science & Business Media, 2007.

[82] Joseph Polden, Zengxi Pan, Nathan Larkin, and Stephen van Duin. Adaptive partial shortcuts: Path optimization for industrial robotics. *Journal of Intelligent & Robotic Systems*, 86(1):35–47, 2017.

[83] Maryam Khazaei Pool, Carlos Diaz Alvarenga, and Marcelo Kallmann. Path smoothing with deterministic shortcuts. In *2022 Sixth IEEE International Conference on Robotic Computing (IRC)*, pages 411–415. IEEE, 2022.

[84] Zhihua Qu, Jing Wang, and Clinton E Plaisted. A new analytical solution to mobile robot trajectory generation in the presence of moving obstacles. *IEEE Transactions on Robotics*, 20(6):978–993, 2004.

[85] Abhijeet Ravankar, Ankit A Ravankar, Yukinori Kobayashi, and Takanori Emaru. Symbiotic navigation in multi-robot systems with remote obstacle knowledge sharing. *Sensors*, 17(7):1581, 2017.

[86] 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.

[87] Volodymyr Savkiv, Roman Mykhailyshyn, Frantisek Duchon, Olegas Prentkovskis, Pavlo Maruschak, and Illia Diahovchenko. Analysis of operational characteristics of pneumatic device of industrial robot for gripping and control of parameters of objects of manipulation. In *TRANSBALTICA XI: Transportation Science and Technology: Proceedings of the International Conference TRANSBALTICA, May 2-3, 2019, Vilnius, Lithuania*, pages 504–510. Springer, 2020.

[88] Thomas W Sederberg. Computer aided geometric design. *Computer Aided Geometric Design*, 2012.

[89] Christoph Siedentop, Robert Heinze, Dietmar Kasper, Gabi Breuel, and Cyrill Stachniss. Path-planning for autonomous parking with dubins curves. In *Uni-DAS Workshop Fahrerassistenzsysteme*. Uni-DAS eV Darmstadt, Germany, 2015.

[90] Igor Škrjanc and Gregor Klančar. Optimal cooperative collision avoidance between multiple robots based on bernstein–bézier curves. *Robotics and Autonomous systems*, 58(1):1–9, 2010.

[91] Alaa Tharwat, Mohamed Elhoseny, Aboul Ella Hassanien, Thomas Gabel, and Arun Kumar. Intelligent bézier curve-based path planning model using chaotic particle swarm optimization algorithm. *Cluster Computing*, 22(2):4745–4766, 2019.

[92] Bailing Tian, Wenru Fan, Rui Su, and Qun Zong. Real-time trajectory and attitude coordination control for reusable launch vehicle in reentry phase. *IEEE Transactions on Industrial Electronics*, 62(3):1639–1650, 2014.

[93] Xiaolong Wang, Alp Sahin, and Subhrajit Bhattacharya. Coordination-free multi-robot path planning for congestion reduction using topological reasoning. *Journal of Intelligent & Robotic Systems*, 108(3):50, 2023.

[94] Wikipedia. Probabilistic roadmap. Retrieved November 12, 2023.

[95] Jun Wu, Guang Yu, Ying Gao, and Liping Wang. Mechatronics modeling and vibration analysis of a 2-dof parallel manipulator in a 5-dof hybrid machine tool. *Mechanism and Machine Theory*, 121:430–445, 2018.

[96] Zicong Wu, Weizhou Su, and Junhui Li. Multi-robot path planning based on improved artificial potential field and b-spline curve optimization. In *2019 Chinese Control Conference (CCC)*, pages 4691–4696. IEEE, 2019.

[97] Hui Yang, Jie Qi, Yongchun Miao, Haixin Sun, and Jianghui Li. A new robot navigation algorithm based on a Double-Layer Ant algorithm and trajectory optimization. *IEEE Transactions on Industrial Electronics*, 66(11):8557–8566, 2018.

[98] Qing Ye, Chaojun Gao, Yao Zhang, Zeyu Sun, Ruochen Wang, and Long Chen. Intelligent vehicle path tracking control method based on curvature optimisation. *Sensors*, 23(10):4719, 2023.

[99] Zhiheng Yuan, Zhengmao Yang, Lingling Lv, and Yanjun Shi. A bi-level path planning algorithm for multi-agv routing problem. *Electronics*, 9(9):1351, 2020.

[100] Xiaojing Zhang, Alexander Liniger, and Francesco Borrelli. Optimization-based collision avoidance. *IEEE Transactions on Control Systems Technology*, 29(3):972–983, 2020.

[101] Zhen Zhang, Defeng Wu, Jiadong Gu, and Fusheng Li. A path-planning strategy for unmanned surface vehicles based on an adaptive hybrid dynamic stepsize and target attractive Force-RRT algorithm. *Journal of Marine Science and Engineering*, 7(5):132, 2019.

[102] Ziang Zhang, Ziyi Zou, Xiang Li, Mingyi Wang, Yixu Wang, Xiaoqing Guan, You Wang, and Guang Li. Path planning for autonomous driving with curvature-considered quadratic optimization. In *2023 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–7. IEEE, 2023.

[103] TI3927120 Zohdi. The game of drones: rapid agent-based machine-learning models for multi-uav path planning. *Computational Mechanics*, 65:217–228, 2020.

[104] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. CHOMP: Covariant Hamiltonian Optimization for Motion Planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.

[105] Uri Zwick. Exact and approximate distances in graphs—a survey. In *European Symposium on Algorithms*, pages 33–48. Springer, 2001.