TRAJECTORY PLANNING AND TRACKING FOR AUTONOMOUS VEHICLES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

HALUK LEVENT ÇIÇEK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2022

Approval of the thesis:

TRAJECTORY PLANNING AND TRACKING FOR AUTONOMOUS VEHICLES

submitted by **HALUK LEVENT ÇIÇEK** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. Klaus Werner Schmidt
Supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Afşar Saranlı
Electrical and Electronics Engineering, METU _____

Prof. Dr. Klaus Werner Schmidt
Electrical and Electronics Engineering, METU _____

Assoc. Prof. Dr. Mustafa Mert Ankaralı
Electrical and Electronics Engineering, METU _____

Assist. Prof. Dr. Ulaş Beldek
Mechatronics Engineering, Çankaya University _____

Assist. Prof. Dr. Halit Ergezer
Mechatronics Engineering, Çankaya University _____

Date: 27.12.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Haluk Levent Çiçek

Signature :

ABSTRACT

TRAJECTORY PLANNING AND TRACKING FOR AUTONOMOUS VEHICLES

Çiçek, Haluk Levent
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Klaus Werner Schmidt

December 2022, 90 pages

Finding appropriate paths is an essential issue for the development of autonomous vehicles and robots. Hereby, it has to be considered that autonomous vehicles cannot follow sharp corners, as they cannot turn on a single point. Therefore, it is important to compute smooth paths that have additional desirable properties such as minimum length and sufficient distance from obstacles. Furthermore, practical applications require the computation of such paths in real time.

This thesis develops a general method for path planning and tracking of autonomous vehicles. In line with the stated requirements, the proposed algorithm ensures smooth curves and avoids sharp corners on the planned path. The proposed algorithm is based on different existing path planning algorithm which named as the Voronoi Boundary Visibilty and Steiner Points Repeatedly (VV-ST-R) that result in straight-line paths with corners. The undesired sharp corners resulting from these algorithm are replaced by smooth curves using Bezier curves. In this way, the traceability of the road is increased. As performance criteria, the path's calculation time, the shortest path distance to the obstacle, and the total length of the path are determined. In various computational experiments, the proposed algorithm and the previous algorithm are

compared. In addition, proposed method for choosing Bezier curve control points are evaluated. It is found that the proposed algorithm results in short smooth paths with a sufficient obstacle distance and a small computation time. Finally, the traceability of the proposed paths is confirmed by driving simulations with car-like robots using the Pure Pursuit algorithm for path tracking. It is further expected that the proposed method can be used for both road vehicles and mobile robots.

Keywords: Path planning, Tracking, Autonomous vehicles, Mobile robots, Bezier Curves

ÖZ

OTONOM ARAÇLAR İÇİN YOL PLANLAMASI VE TAKİBİ

Çiçek, Haluk Levent
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Klaus Werner Schmidt

Aralık 2022 , 90 sayfa

Uygun yolları bulmak, otonom araçların ve robotların geliştirilmesi için önemli bir konudur. Burada otonom araçların tek bir noktadan dönemedikleri için keskin virajları takip edemeyecekleri de göz önünde bulundurulmalıdır. Bu nedenle, minimum uzunluk ve engellerden yeterli mesafe gibi ilave istenen özelliklere sahip düzgün yolları hesaplamak önemlidir. Ayrıca, pratik uygulamalar bu tür yolların gerçek zamanlı olarak hesaplanmasını gerektirir.

Bu tez, otonom araçların yol planlaması ve takibi için genel bir yöntem geliştirmektedir. Belirtilen gereksinimler doğrultusunda, önerilen algoritma düzgün eğriler sağlar ve planlanan yolda keskin köşelerden kaçınır. Önerilen algoritma, Voronoi Boundary Visibilty and Steiner Points Repeatedly (VV-ST-R) algoritması ile türetilen köşeli düz çizgi yollar ile sonuçlanan mevcut yol planlama algoritmasına dayanmaktadır. Bu algoritmadan kaynaklanan istenmeyen keskin köşeler, Bezier eğrileri kullanılarak düzgün eğrilerle değiştirilir. Bu sayede yolun takip edilirliği artırılmış olur. Performans kriteri olarak yolun hesaplama süresi, engele en kısa yol mesafesi ve yolun toplam uzunluğu belirlenmiştir. Çeşitli hesaplama deneylerinde önerilen algoritma ve önceki

algoritmalar karşılaştırılır. Ek olarak, Bezier eğrisi kontrol noktalarının seçilmesi için önerilen yöntem değerlendirilir. Önerilen algoritmanın, yeterli bir engel mesafesi ve küçük bir hesaplama süresi ile kısa düzgün yollar ile sonuçlandığı bulunmuştur. Son olarak, önerilen yolların izlenebilirliği, yol takibi için Pure Pursuit algoritması kullanılarak araba benzeri robotlarla simülasyonlar yapılarak doğrulanır. Ayrıca önerilen yöntemin hem karayolu taşıtları hem de mobil robotlar için kullanılabilmesi beklenmektedir.

Anahtar Kelimeler: Yol planlama, Yol takibi, Otonom araçlar, Mobil robotlar, Bezier Eğrileri

To my beloved ones...

ACKNOWLEDGMENTS

I would like to thank my supervisor Prof. Dr. Klaus Werner Schmidt for his patience, support and assistance during this period.

I sincerely thank to my family and my friends for their support during the thesis work.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
MPC	Model Predictive Control
LiDAR	Light Detection and Ranging
NDT	Normal Distribution Transform
APF	Artificial Potential Field
IMU	Inertial Measurement Unit
PRM	Probabilistic Roadmap
GA	Genetic Algorithm
GVD	Generalized Voronoi Diagram
VV	Voronoi Boundary Visibility
VV-ST	Voronoi Boundary Visibilty and Steiner Points
VV-ST-R	Voronoi Boundary Visibilty and Steiner Points Repeatedly

CHAPTER 1

BACKGROUND INFORMATION

1.1 Introduction

Autonomous driving technologies have been researched for years. DERVISH, an office navigation robot, won a robot competition in 1994 [1]. Many mobile robots and autonomous vehicles have been developed since then. There are many differences in these autonomous vehicles produced throughout history. As technology evolves and new studies are carried out in this field, many car features have changed. However, there are also standard features. One of these common features is the path planning feature. The movements of these autonomous vehicles are not random, and they somehow plan their paths and follow that path to accomplish tasks. This thesis was written on the path planning and tracking of autonomous vehicles and mobile robots. The path planner aims to create a safe path between the start and end points without hitting obstacles in the vehicle's environment. Following this generated path properly is provided by the motion controller [2]. There are many studies in the literature for path planning. These studies were conducted on some features considered necessary in path planning. Path length, the time required to find the map, and safety can be considered fundamental problems to be solved by path planning studies [3]. However, most of these studies do not offer a comprehensive solution to these problems. For this reason, there is a research [4] to explain and compare these solution proposals in detail. After this comparison, according to the purpose, suitable algorithm can be selected.

On the one hand, there are algorithms that directly compute smooth paths using certain types of curves such as splines, polynomial spiral, arcs, clothoids and Bezier

curves [5]. Such methods may have the disadvantage of larger computation times in some cases [6]. On the other hand, there are methods that smoothen previously computed simple paths with corners. For example, [7] tries to smoothen paths that are constructed from hypercloids, [8] tries to smoothen paths together with a limit on the maximum curvature and [9] introduces a parametric continuity in order to increase the traceability of paths.

1.2 Basic Concepts

Some concepts need to be known in order to understand autonomous motion fully. Autonomous motion is a topic that comprises multiple research fields. In order to realize fully autonomous motion, it is necessary to bring together many different concepts. Although the main topic of this thesis is path planning and tracking for autonomous vehicles, it is essential to have information about other concepts used in the autonomous movement. These basic concepts can be specified as localization, mapping, perception, planning and waypoints, and motion control.

1.2.1 Localization

Knowing where the autonomous robot is in its environment is one of the essential features of autonomous movement. The autonomous robot must reach the goal position while moving in the environment for autonomous movement.To do this, it must know the start and goal point. However, just knowing these is not enough for successful autonomous motion. The robot must also know its position and whether it has reached the start or goal point. Even if the autonomous robot localizes itself in its immediate vicinity after not knowing its global position in the environment, this is not enough for autonomous driving [10]. In [10], detailed research on localization for autonomous motion has been done. The following two techniques Satellite Navigation Systems and Landmark-Based Navigation, are two popular techniques for localization for autonomous motion.

- Satellite Navigation Systems

  The quality of the localization may vary depending on the navigation system used. Generally, the information from the navigation system is used by combining it with the digital map. A Digital map provides the information necessary for the autonomous robot to detect the road. However, only using navigation systems is not a satisfactory solution for localization. First, for using navigation systems, open space is required. If buildings or underpasses are affecting the signals, the results from the navigation systems will not be of the desired quality. That is why sensors are used in the vehicle along with navigation systems [10].

- Landmark Based Navigation

  Landmarks are unique structures, and these landmarks can be detected during autonomous driving with the help of vehicle sensors. Since landmarks are unique, when sensors detect these landmarks, relative position can be known according to the measurements. After that, the global positions can easily be found. Lidar and other vision sensors can be used to detect these landmarks [10].

1.2.2 Mapping and Perception

An autonomous robot needs a map to localize itself inside that map. For the autonomous robot to localize, it needs the data obtained from the environment and the map data. In other words, the vehicle should know where the obstacles are and where the accessible areas are. This map can be built with sensor measurements. When an autonomous robot localizes itself, its measurements are the information about its vicinity. As the vehicle moves, this information is acquired in different parts of the map. The main phases of this operation are perception, processing, and fusion.

Perception is the result of sensor measurements. At each motion sequence, different sensor readings can be gathered. Processing comes after perception to build a local map representation of the vehicle's vicinity. This local representation is transferred to global representation in the fusion phase [11] .

1.2.3 Planning

The primary purpose of this thesis is path planning and tracking for autonomous robots. More detailed explanations and studies in this field will be provided later in detail. However, it is necessary to give a summary before going on to the studies in this field in the literature. Every autonomous movement has a specific purpose. In its most general definition, the robot should be able to go from where it starts to move to where it will arrive without hitting obstacles. Along this path, the autonomous robot must not hit obstacles and must not get lost. For these reasons, autonomous movement does not depend on a single concept. All concepts such as localization, mapping, and perception mentioned in previous sections have the same importance simultaneously.

Since this thesis focuses on path planning, the following sections will focus on this and give detailed information about that. Each path planning algorithm evaluates different parameters. Each algorithm has different features, such as serving different autonomous robot models. An algorithm that works for one autonomous robot may not be suitable for another. However, all these algorithms also have some standard features. These properties are used to indicate and compare the performance of the planned path. The main parameters of path planning are safety, path length, and computation time. [12]. Algorithms in this field can be compared with each other with these parameters.

1.2.4 Waypoints

Waypoints are points located at specific points that make up the planned path the vehicle is intended to follow. In many studies in path planning, a series of waypoints are used to indicate paths. Waypoints are selected from the points where there are no obstacles. In order to be a safety path, if there is an obstacle on the path, algorithms aim to find a path without hitting the obstacle by generating more waypoints. From the starting point to the ending point, the autonomous robot is guided by waypoints in this way. In path tracking studies, research is carried out on the controllers that can follow these waypoints [13]. The connection of these waypoints with each other is

directly related to the vehicle to be used in the study. Detailed information about this topic will be given in chapter 3.

1.2.5 Motion Control

The autonomous robot finds a possible path to reach the goal with the help of path planning algorithms. After the robot's path is planned, it needs a controller to follow this path. At that point, the motion control problem occurs. Motion control of autonomous robots can be divided into three sub-problems: stabilization to a point, path tracking, and path following. Path tracking and path following can be considered interrelated. The difference between these two problems is their dependency. Path tracking is a function of time. On the other hand, path following depends on the vehicle's current state or different data [14]. During this tracking operation, a steady motion is essential. Although a path has been created between the start and end points, it is also essential to follow this path steadily. If this tracking is not steady, hitting the obstacles around the planned path may occur. Even if there is no obstacle, steady tracking is preferred for the comfort of the journey.

1.3 Path Planning and Tracking

One of the essential factors in path planning and tracking studies is whether the environment is dynamic or static. In dynamic environments, the planned path should be changed when an obstacle, person, or object is not expected to be there. This change must be made quickly and safely. The robot may harm itself, its surroundings, and even living beings in a possible collision. Therefore, the parameters to be considered in dynamic environments are higher than in static environments. In addition, the reaction time to environmental changes is as important as the planning of the path. In static environments, there will not be such a problem as any unexpected obstacle is not expected on the pre-planned path. However, since dynamic environments should be included in possible situations in a realistic study, studies were carried out in both scenarios. However, there are some standard features for the generated paths in both scenarios. Some features are listed below.

- Obstacle free paths
- Less computation time
- Paths as short as possible and with smooth turns
- Drivable paths that match the vehicle's dynamic model

  The study's results will not be efficient if the paths generated are the paths the vehicle cannot follow in a realistic scenario. This feature is also related to the path having smooth turns. For instance, a car-type autonomous robot cannot follow sharp turns with straight lines. Therefore, it is essential that the path found for this type of vehicle has smooth turns. On the other hand, omnidirectional robots that can turn in place can turn such sharp turns [3]. Thus, even if the turns on the path are smooth, an undesirable situation may occur if the road is long in this type of vehicle.

1.4 Related Works

1.4.1 Algorithms Used for Path Planning and Tracking

As mentioned above, finding a path between the starting and ending positions is not enough without hitting obstacles on the path. It is also an essential factor that the path length. Besides, it is also important in the path has smooth turns. Studies in the field of path planning and tracking focused on these parameters and compared different algorithms. In [15], researchers compared Probabilistic Roadmap (PRM) and Genetic Algorithm (GA) techniques. As a result of their studies, both techniques gave the desired result. However, they also have advantages and disadvantages against each other. Although PRM has less computation time than GA, GA has been found to find paths with smoother turns compared to PRM. For this reason, the better approach will change according to the usage area.

In addition to static environments, path planning and tracking studies in dynamic environments were also examined to contribute to the path planning and tracking problem. The solution proposal has been published in [16] path planning and tracking in dynamic environments problem. They used the sigmoid function to find a reference

path that can be used to respond to changes in dynamic environments, and their solution is based on Model Predictive Control (MPC). A sigmoid function can be defined as a function that has an S-Shaped curve. With the help of the sigmoid function, they establish a reference trajectory. However, this reference trajectory should be changed according to the obstacle's position in dynamic environments. They use lidar to detect the obstacle's position, and they define this position by using lidar point clouds. They assume that all objects can move with speed and acceleration. When the obstacle is detected, they use the MPC's predictive horizon state by using the obstacles' coordinates, speed, and acceleration to predict the position. After that, they publish a risk index, a cost function to optimize collision avoidance during the tracking. Then, their algorithm adapts to dynamic changes in the environment and chooses the optimal scenario.

Research on path planning and tracking are also carried out within the scope of autonomous overtaking. In [17], the authors reviewd of studies in this area. Since autonomous overtaking will be in a dynamic environment, they drew attention to what issues should be considered while working accordingly. As a result of their work, they published the following information. They mentioned the importance of considering the vehicle's dynamics, the environment's constraints, and other information about the environment for autonomous overtaking. Since the environment is dynamic, highly accurate environment information is needed during autonomous lane changing. The vehicle can also access other environmental information with the Vehicle to Everything communication. Thanks to this communication, the perception range will be expanded, and the accuracy of the data will be increased.

In order for the studies to be efficient, autonomous driving must be accepted by the people. To increase this, there are studies in the literature by observing human behaviors. There are examples in the research conducted in this area, considering human behavior. The subsequent two studies contributed to this field by examining human behavior. Many types of path planning and tracking research start with safety criteria since it comes first in path planning applications Some studies start by examining human drivers to offer adequate solutions for safety [18]. They start their studies by analyzing the habits of a human driver. They aim to develop a safety model for obstacle avoidance. Then, they use this safety model to build the artificial potential field

(APF). Then, their algorithm finds a feasible path using this artificial potential field.

The acceptability of autonomous vehicles can be increased by designing a controller with human behaviors during traffic scenarios. In [19], the authors aim that in their study. They first modeled people's driving styles in the traffic environment using Artificial Potential Field. Then, they used this data in controller design processes to optimize paths and control outputs. Thus, they enabled the autonomous vehicle to act in human habits in traffic scenarios. The experiments in this study were conducted in two modes: lane changing and car following scenarios. In these experiments, they have seen that the controller reflects human behaviors using both careful driver and aggressive driver models.

1.4.2 Algorithms Used for Smooth Path Planning and Tracking

The primary purpose of this thesis is not only path planning and tracking for autonomous robots but also aims to include smooth curves instead of sharp corners as much as possible. Some smooth curve generation studies that have been found as a result of research that can be done are described below.

A review article also described the advantages and disadvantages of path smoothing techniques. In [9], the advantages and disadvantages of splines are also mentioned. Having a low computational cost of splines is a reason for preference. Using splines allows using continuity on the path. However, the difficulty of creating a trade-off between continuity and desired shape when using splines is also mentioned.

In [9], hypocycloids are the other technique whose advantages are described. It is mentioned that they will be preferred because they are easy to calculate, but they also have disadvantages as they do not guarantee continuity.

Studies in this field can be examined under two main headings. The first one is algorithms that directly compute smooth paths. The second one is algorithms that smoothen paths with corners.

1.4.2.1 Algorithms that Directly Compute Smooth Paths

It will be helpful to examine more than one technique with a study that considers the algorithms used to smooth the path in the field of path planning as a survey. In [5], they mentioned Bezier Curves, Spline Curves, Polynomial Spirals, Arcs, and Clothoids to find a smooth curve during path planning. Each technique has both advantages and disadvantages. Any of these techniques can be used according to the usage area. Control points are used in the Bezier curve. The first and last control points are the curve's beginning and end. Waypoints, usually in the middle, cannot be included in the path. They have a global effect in creating the shape of the path.

In [20], B-spline curves are used to generate smooth paths. Their algorithms take the positions of obstacles in the environment, the environment's boundaries, and the vehicle's positions at the start and end points. It then aims to create a smooth path between the start and end points by creating a B-spline curve such that curvature is continuous with upper bounded curvature. There is a small slope discontinuity in the knots of this curve created. However, since adjusting upper bounded curvature constraints on narrow roads is challenging, their algorithm is unsuitable for use in these areas.

In [6], they use Bezier curves to generate smooth paths. They also improved the positioning of control points, making the Bezier curve challenging. In their algorithm, they first create an n-th order Bezier curve and use this curve as a base. This curve has location and velocity endpoint constraints. Then the curve is created. They modified their algorithm for a path to be created between this base curve and control polygons. They define the point they want the road to pass over as a via-point. Then they determine the closest point on this via-pointe base curve. Then, they give the path its final shape by applying a step that allows the path to passing through the via-point with these defined points. In obstacle scenarios, they aim for the vehicle to follow the modified path by positioning via points in free space. However, they did not conduct a study on obstacle avoidance in their research. Considering parameter such as path length, where to position via points can be a complex problem.

1.4.2.2 Algorithms that Smoothen Paths with Corners

As examples are given above, many studies have been carried out in path planning. These algorithms try to provide the safety criterion by standing at a certain distance from the obstacles. However, these algorithms may not have smooth curves for vehicles to rotate. Therefore, by adding on the previous algorithms, a smooth path is obtained by adding studies. In [7], the sharp turns made in the path planning algorithms using hypercloids were removed, and soft turns were added. Their algorithm are adaptable to commonly used algorithms such as probabilistic roadmap or A*. In addition, these algorithms are focused on replacing only sharp turns with smooth curves. It does not include a solution for straight-line road segments without sharp turns. In addition, they also considered the presence of more than one robot in the environment and collision with each other in their studies. To prevent this situation, they generate nodes.

[8] offers different path smoothing techniques. Upper-bounded continuous curvature path-smoothing algorithm was chosen as the basic algorithm of their algorithms. In the first algorithm example, they softened the path by using the empty spaces formed in the linear paths. They used the maximum curvature feature as the second solution proposal. When the maximum curvature constraint is not provided, the trackability of the path will decrease, so this algorithm is used. They shorten the distance required to provide the maximum distance constraint with their proposed algorithm. When they were tested by combining the proposed algorithms, they both smoothed the path and provided the maximum curvature constraint.

It will not be satisfactory enough that the path found is smooth. Having certain features of the created smooth path will increase its preferability. In [9], they conveyed the features that increase the preferability of path continuity. They explained two different types of continuity geometric and parametric continuity. Parametric continuity can be defined as both the smoothness of the path and its parametrization. Geometric continuity can be defined as the quality of the curve that the robot will follow. While discontinuity between curves prevents paths from being preferred, tangential discontinuity and curvature discontinuity are also not preferred conditions. The maximum curvature concept can be used for curve traceability. The smaller this value, the higher

the traceability of the path will be.

Pure pursuit algorithm is one of the methods used for path tracking as it is a very successful algorithm in the field of path tracking. The pure pursuit algorithm is called the geometric path tracking algorithm [21]. The algorithm aims to be on the pre-planned path producing and using the steering angle. The algorithm first creates a look-ahead point based on the vehicle's current location and the look-ahead distance. The algorithm creates a steering angle with this information and sends it to the autonomous robot. After the autonomous robot fulfills the incoming command, it sends the current position information again, and the algorithm continues this way.

1.4.3 Bezier Curve Usage for Smooth Path Planning

There are some smooth path planning methods such as; line and arc, spline curve, clothoid curve and Bezier curve. Along these methods spline curve, clothoid curve and Bezier curve methods have curvature continuity. If the path complexity feature is compared along these methods, Bezier curve's path complexity is simpler [22] than other methods. Thus, following articles prefer to use Bezier curves to generate smooth paths.

Comparison parameters of path planning algorithms are the shortest distance, lower computation time, and completeness. For the car-like robots kinematic constraints must be considered while planning path, such that to guarantee that the robot can reach goal position. Using Bezier curves tries to find feasible path such that kinematic constraints are considered. However, selection of the control points is one of the main issues using Bezier curves. Different works have been done about how to select Bezier curve control points.

In [23], there is a pre-planned piecewise linear path and Bezier curves are derived from this path. By doing this, they aimed to make pre-planned path feasible and trackable by car-like robots with kinematic constraints. Since the piecewise linear path include sharp corners, and vecihcle fails to follow that path, they introduce new method called Variable Waypoint Offset (VWO). Their offset value is proportional to the heading angle of the vehicle and vehicle's speed. From two adjacent line segments

of pre-planend path, they calculate this offset value by using minimum turning radius and heading angle. By using this offset value, they create a circle along the waypoint and selects control points on that circle.

In [22], authors describe a strategy to find an obstacle free path between start and goal position while tracking with good performance and smooth. Their work uses on-line Bezier curves to find a smooth path while the robot faces an obstacle during its movement. In their work, fourth order Bezier curve is used and first and last control points are start and end point of the path, respectively. Their algorithm generates two new control points which close to the start and end points. A quarter distance between start and goal points and these two points are generated at this quarter distance away from start and end points. To generate last control point, their algorithm uses desired curvature value. With desired continuous curvature, new point is generated on the vehicle's moving direction. If resulting Bezier curve hits obstacle, algorithm generates a new intermediate goal point and finds obstacle free path.

In [24], authors consider curvature continuous paths by generating feasible path for autonomous robots. Since Bezier curve with large number of control points' computation time is larger and numerically unstable. They used low degree Bezier curves to find a smooth path. The first step of their algorithm is segmentation of the map such as straight segments and corner segments. Along the map, they used different Bezier curves segments with continuous at their end points. After segmentation, they used quadratic Bezier curves at the corner segments. At the straight segments, they used a Bezier curve according to the minimum number (minimum six) of control points required for continuity.

There are some works Voronoi diagrams and Bezier curves to find a smooth path for autonomous robots. Since the proposed method of this thesis is similar to this procedure, it will be better to explain their work. The following two work uses Voronoi diagrams and Bezier curves.

In [25], piecewise linear path which is obtained from Voronoi diagram is used with collision free Bezier curves to find the shortest and smooth path. Since the resulting curve stays between convex hull of control points, they used Bezier curve to guarantee that obstacle free path. In first step of their algorithm, they add some more points into

a path so that the distance between them is more than a certain amount. These new points is also helpful for the implementation of the Bezier curve in the sections where the path is divided into parts in the algorithm.

In [26], Bezier curve based on Voronoi diagram is used to generate smooth paths. Their work is based on four steps. In the first step, piecewise linear path is found along the Voronoi Diagram. Then, Dijkstra algorithm is used to find the shortest path. As a result, waypoints are located away from the obstacles to find an obstacle free path, and these waypoints are used as control points for Bezier curve in the next steps. Since Voronoi Diagram creates multiple waypoints to find an obstacle free path in the corners, there are some crowded waypoints along the path. As a second step, their algorithm removes these crowded control points to reduce the curve length. In the third step, control points are subdivided into two ordered subsequences to obtain collision free Bezier curve. As a final step, they add some additional control points for smoother Bezier curve. Therefore, the resulting paths meet kinematic constraints for autonomous robots. Then, as a next step they remove crowded control points to reduce the curvature and path length. Finally, to find a continuously differentiable Bezier curve segments, they insert additional control points. After all of these preperations, they build Bezier curve along the path.

1.4.4 Advantages and Disadvantages of Smooth Path Planning Algorithms

In this section advantages and disadvantages of smooth path planning algorithms are given [9].

Interpolation based smooth path planning algorithms are easy to compute, and curves can be concatenated to get desired smooth path shapes. Also, they can be used for local path planning for safety. However, when they are used it is difficult to control coefficients of curves of higher order (>4), at higher order their computation time becomes large and they are not suitable for high speeds .

Splines have low computational costs and provide curvature continuity which is important. However, when splines are used there is a trade-off between continuity and shape, and it becomes difficult to balance it.

Another method is Dubin's curves to generate smooth paths. Dubin's curves can be used because they are easy to compute and computation time is fast. However, there is no curvature continuity.

Optimization based methods takes into account different constraints, and they can be combined with other approaches. However, these methods consumes too much time and might not necessarily converge.

Bezier curves is used to obtained smooth paths for this thesis. Bezier curves have low computational cost, and control points allow to generate desired shape of path. Also, multiple Bezier curves can be connected to each other to get desired path shape. However, computation time increase at high orders. Since control points affects entire path, it is difficult to place these control point's locations. In chapter 3, the proposed method is given to solve these problems while generating smooth paths.

1.5 Proposed Methods and Models

The main goal in this thesis is the computation of smooth paths for autonomous robots in real time. To this end, our proposed method uses Bezier curves to generate smooth curves instead of sharp corners in the planned path. We are using existing waypoints generated from VV-ST-R that compute straight-line paths to add Bezier curves to the path planning algorithms. Specifically, we are using these waypoints as control points of Bezier curves.

This thesis will explain the proposed algorithm to generate smooth paths between start and goal positions. The proposed algorithm uses Bezier curves to generate smooth curves. A detailed explanation of Bezier curves and the proposed algorithm will be given in chapter 3 and chapter 4.

1.6 Contributions and Novelties

Our main contributions are as follows:

- Development of an algorithm for refining straight-line paths by adding Bezier

curves to remove corners,

- Avoiding obstacles by placing control points and waypoints in the obstacle-free space,
- Handling curvature constraints,
- Application of the proposed method with different obstacle maps for both car-like robots with Ackermann steering and differential drive robots.

1.7 The Outline of the Thesis

The remainder of the thesis is as follows.In chapter 2, motivation is given. Then, basis studies of the proposed method will be examined from previous research. In chapter 3 The Bezier curve technique, which generates smooth curves instead of sharp corners, will be explained. Then, proposed algorithms to generate Bezier curves, will be explained. In chapter 4 our proposed solution will be examined. Experimental results will be shown. The effects of using Bezier curves and comparisons with different test scenarios will be explained. Finally, the traceability of the path will be checked by Pure Pursuit algorithm.

CHAPTER 2

MOTIVATION

The main focus of this thesis is the computation of paths for non-holonomic robots such as differential-drive robots or car-like robots with Ackeremann steering. In this context, general performance criteria of path planning algorithms are path length, computation time and finding obstacle-free paths. In addition, it is important to determine drivable paths that can be tracked by the robots. This chapter provides the motivation for the path-planning method developed in this thesis. Section 2.1 introduces the generalized Voronoi diagram and Section 2.2 outlines the sampling-based straight-line path planning method that is used as the basis for the developed method. Then, Section 2.3 outlines the main idea of this thesis work with the aim of computing drivable paths for differential-drive and car-like robots.

2.1 Generalized Voronoi Diagram

Voronoi diagrams have wide usage area. In archaeology, biology, computer science etc. Voronoi diagrams are used because of their properties. If there are (N) points on the plane, Voronoi diagrams divide the plane into (N) cells, such that each cell contains only one point. The boundaries of each cell are equidistant from two or more points. All the points inside each cell are closer to the point in the cell than the points of the other cells [27].

The Generalized Voronoi Diagram (GVD) is frequently used for robotic path planning. The main idea of the GVD is to generate a path that keeps the maximum distance to all the surrounding obstacles. To this end, we consider that a robot is operating in an environment with several obstacles as illustrated in Figure 2.1.

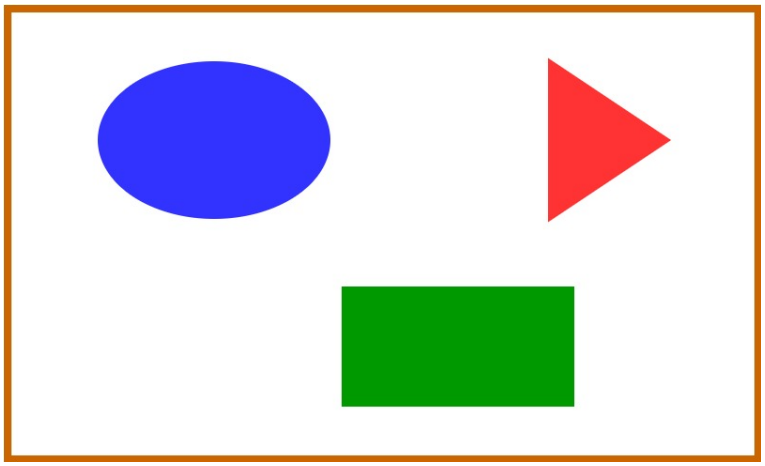

Figure 2.1: Example for an obstacle map with three obstacles and a boundary.

Then, the first step is to separate such map into Voronoi regions. Each Voronoi region contains all the points that are closer to one of the obstacles than to any other obstacle. The Voronoi regions for the obstacle map in Figure 2.1 are shaded in the same color as the respective obstacle in Figure 2.2. Specifically, it can be seen that there are four Voronoi regions. Three of the Voronoi regions belong to the obstacles with different shapes that are placed inside the map area. The fourth Voronoi region belongs to the boundary of the obstacle map.

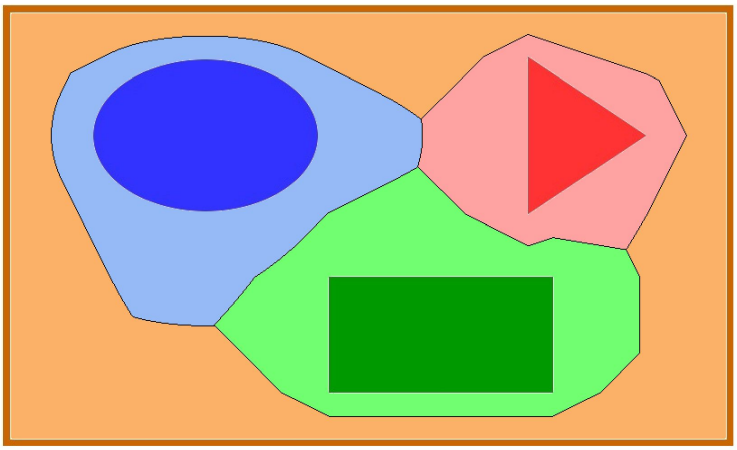

Figure 2.2: Example VV-ST-R Path

It can be seen in Figure 2.2 that the Voronoi regions are separated by a boundary

line. This boundary line has the property that all the points on the boundary line have the same distance to at least two of the obstacles. Furthermore, there are branching points on the boundary line, which correspond to points that have the same distance to more than two obstacles. The defined boundary line is denoted as the obstacle map's Voronoi Boundary (VB). The VB for the obstacle map in Figure 2.1 is shown in Figure 2.3.

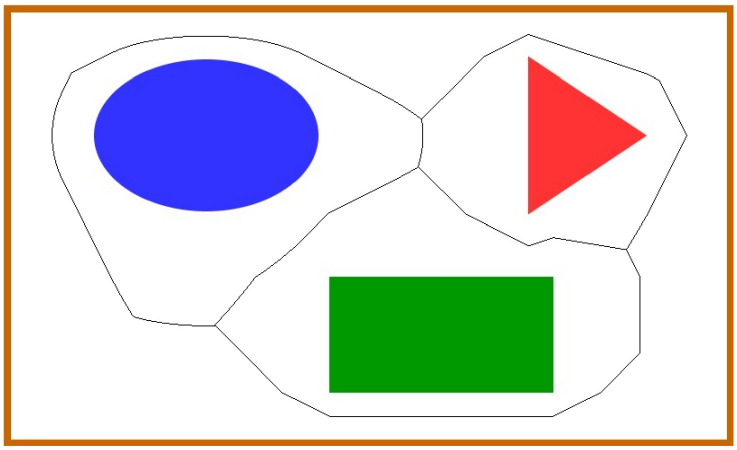

Figure 2.3: Voronoi Boundary of the example obstacle map.

The most important property of the VB is that it characterizes all the points that have a maximum distance to the obstacles in the obstacle map. That is, a robot that follows the VB will keep the maximum obstacle distance and hence travel on a safe path [3]. Nevertheless, it is the case that paths that are defined by the VB can take unnecessary turns and are generally unnecessarily long. In addition, it is not ensured that nonholonomic robots such as differential drive robots or car-like robots can track the VB.

2.2 Voronoi Boundary Visibility (VV) and Steiner Points

As explained before, the points on the VB are at a maximum distance from obstacles. Accordingly, there can be unnecessary turns and long distances when directly following the VB. In order to address this issue, the recent work in [3] develops the

VV_ST_R algorithm that makes use of the VB while shortening the path between a given start and goal position. This algorithm iteratively applies two main steps. As the first step, unnecessary waypoints are removed if a straight-line connection between waypoints is possible without hitting an obstacle. As the second step, additional waypoints are introduced if it is required to cut corners, for example when turning around a curved obstacle. Illustrative examples are shown in Figure 2.4 to 2.7. Here, obstacles are represented by black shapes and are extended by a user-defined safety margin that is depicted in orange. Furthermore, each of the figures shows a candidate path along the VB (magenta) from the start position (red dot) to the goal position (green dot). In addition, the figures show the solution path computed by the VV_ST_R algorithm in [3] (blue).

It can be seen from Figure 2.4 to 2.6 that the resulting paths can tightly follow curved obstacles, while straight-line paths are used to take turns around rectangular obstacles.

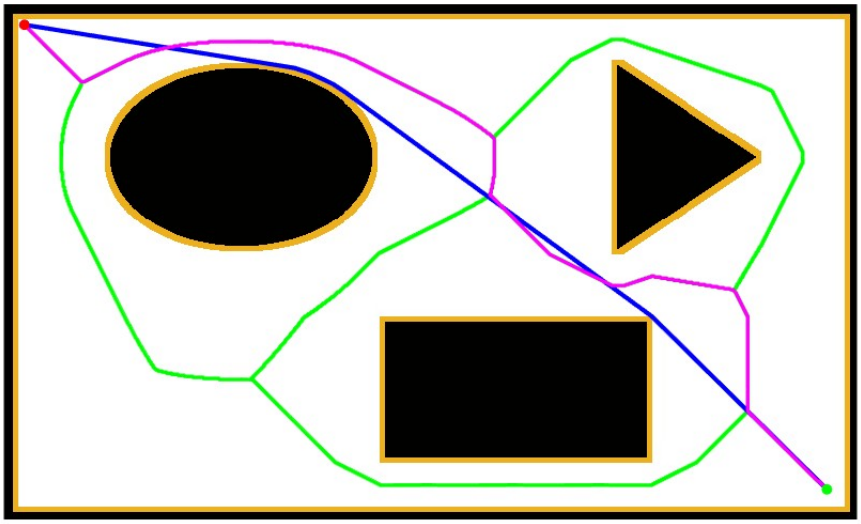

Figure 2.4: Shortest path computation: Path 1.

Moreover, Figure 2.7 gives an example of a solution path that only encounters obstacles in the form of polygons. In this case, the solution path very tightly approximates the shortest straight-line path which can be obtained from the visibility graph as shown in [3].

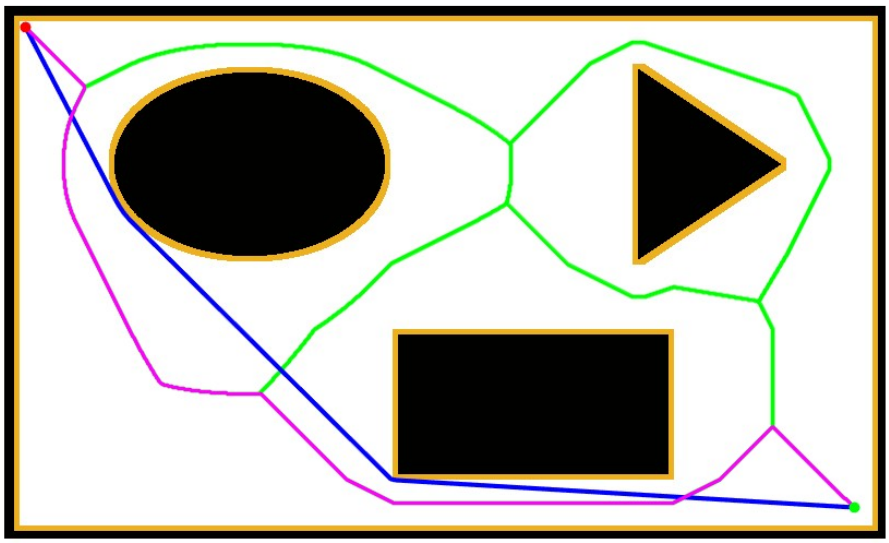

Figure 2.5: Shortest path computation: Path 2.

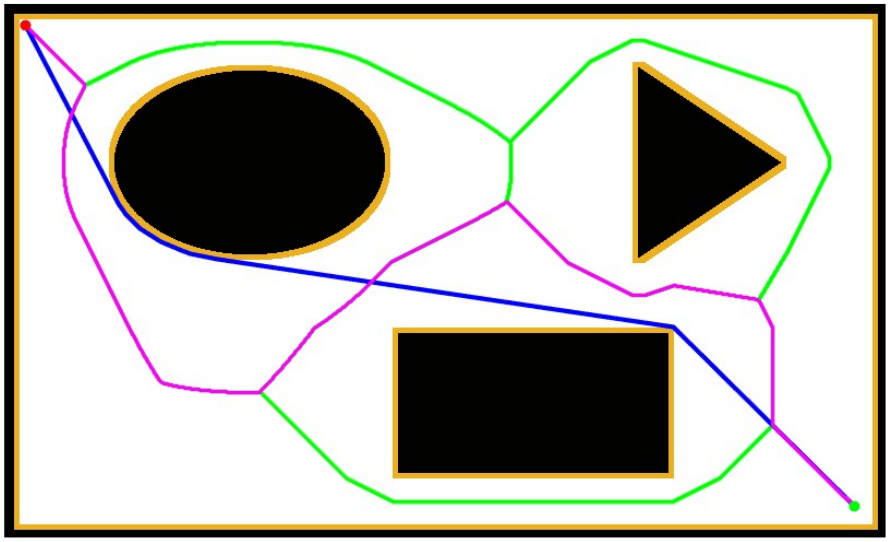

Figure 2.6: Shortest path computation: Path 3.

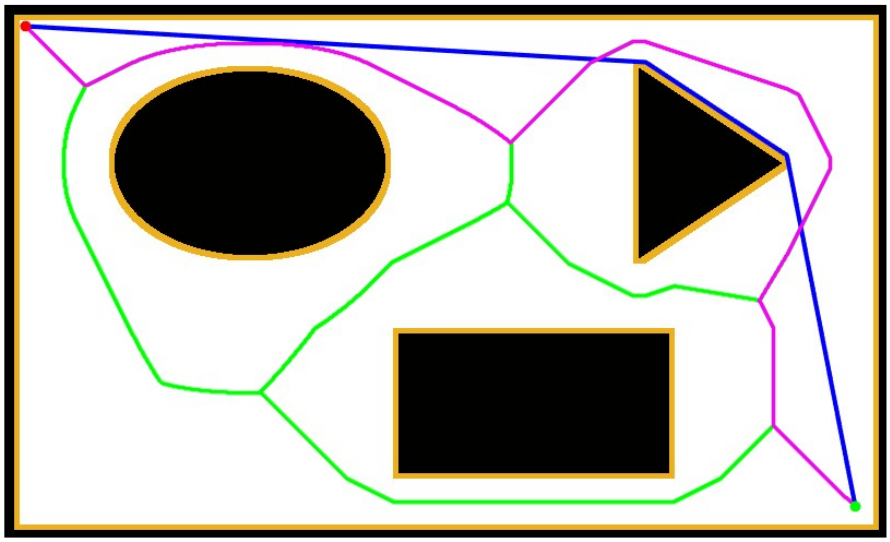

Figure 2.7: Shortest path computation: Path 4.

The stated properties of the solution paths computed with the VV_ST_R algorithm can be further seen in Figure 2.8 and 2.9. Specifically, Figure 2.8 illustrates the properties of solution paths when turning around polygon-shaped obstacles. Similar to a visibility graph, few waypoints are placed at the corners of the obstacles in order to obtain a small number of straight-line segments. Differently, a larger number of waypoints is introduced in the case of curved obstacles as can be seen in Figure 2.9 in order to approximate the curved shape of the obstacle.

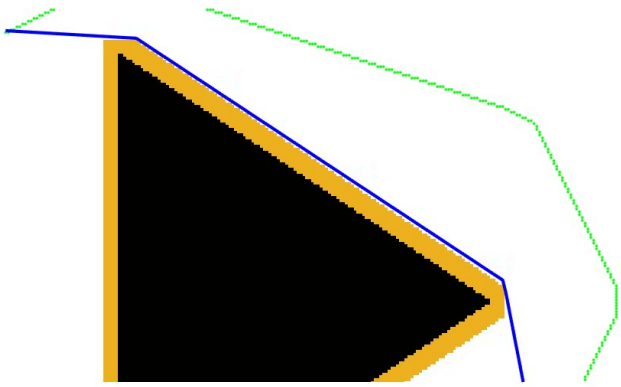

Figure 2.8: Close-up of the solution path around the triangular obstacle in Figure 2.7.

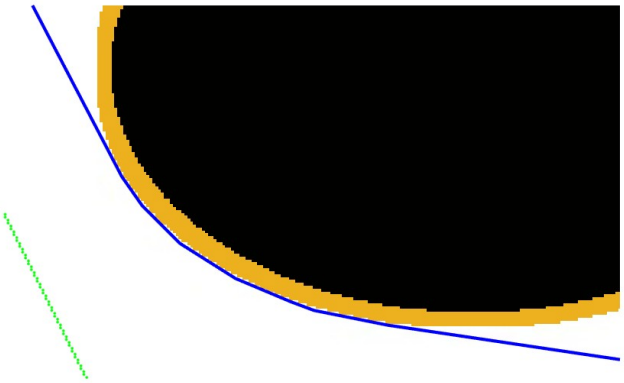

Figure 2.9: Close-up of the solution path around the elliptical obstacle in Figure 2.6.

2.3 Main Idea to Satisfy Drivable Short Paths

As an outcome of the previous discussion, it can be concluded that paths along the VB are safe but generally unnecessarily long. In addition, while paths that are computed by the VV_ST_R algorithms stay at a safe distance from obstacles by introducing a safety margin, such paths are defined by waypoints that are connected by straight-line segments. That is, although such paths can for example be tracked by omnidirectional robots, they are not suitable for nonholonomic robots such as differential-drive robots or car-like robots with Ackermann steering.

The main contribution of this thesis is the computation of smooth paths for nonholonomic robots based on information from paths that are computed by the VV_ST_R algorithm. Hereby, we consider the following important facts:

- In order to obtain safe smooth paths, it is desired to move waypoints away from the obstacles,
- It is known that the area between the solution path and the corresponding VB is obstacle-free space,
- In order to obtain smooth paths, a suitable curve that can easily be parametrized has to be chosen.

In this thesis, the waypoints computed by the VV-ST-R algorithm are used to deter-

mine control points of Bezier curves in order to generate safe and smooth paths to be tracked by nonholonomic robots.

CHAPTER 3

METHOD

3.1 Bezier Curves

In this thesis, Bezier curves are used to generate smooth curves instead of sharp corners along the path. Straight lines connecting the waypoints used during path planning may contain sharp corners for vehicles to follow. Therefore, it is preferable to find paths in the form of smooth curves that vehicles can follow while following waypoints. Bezier curves are one of the methods that can be used for this purpose [6].

Bezier curves can be created with given $n + 1$ control points. The straight lines connecting these control points are called control polygons. The resulting smooth Bezier curve will be formed within this control polygon [28]. In $N - dimensional$ space, the $n - th$ order Bezier curve can be defined as:

$$P(t) = \sum_{i=0}^{n} B_i^n(t) \cdot P_i, (0 \leq t \leq 1), P_i \in R^N \tag{3.1}$$

where B(t) is called Bernstein polynomial and:

$$B_i^n(t) = \binom{n}{i} \cdot t^i (1-t)^{n-i}, \binom{n}{i} = \frac{n!}{i! \cdot (n-i)!} \tag{3.2}$$

Bezier curves have low computational costs, and the curve can be adjusted to the desired characteristic thanks to control points. Moreover, multiple Bezier curves used to obtain a whole path can be combined to create the desired path. On the other hand, as the degree of the Bezier curve increases, the computation cost increases as well, and the adjustment of the curve becomes more difficult. This is because each control point affects the global path [9].

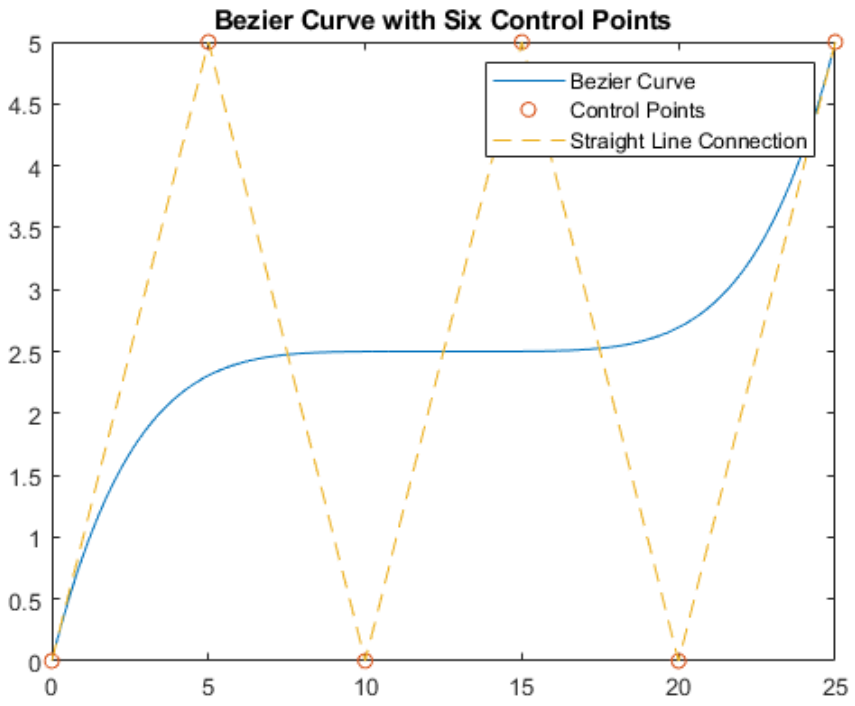

Figure 3.1: Bezier Curve Example with Six Control Points

In Fig. 3.1, an example Bezier curve with six control points can be seen. The solid line curve represents the Bezier Curve and, the dashed lines represent a straight-line connection between control points. As seen in the Fig. 3.1, a smooth curve is defined by the control points. The primary purpose of using Bezier curves is to add some smooth curves on the sharp corners in the planned path to track by the autonomous vehicle. In this thesis Bezier curve of order six is used. The control points' locations have a direct effect on the result curve. Thus, the locations of the each control point must be selected carefully. Since in the path planning algorithms aims that finding obstacle free paths, the locations of the control points needs to be selected to find an obstacle free Bezier curves. Finding the control points locations of the Bezier curves will be examined in Section 3.3.

3.2 Bezier Curves and Curvature

It has to be considered that mobile robots have a lower bound on their turning radius. Considering that the curvature of a path is defined as the inverse of the turning radius, this implies that the curvature of paths for mobile robots needs to be limited. That is, when using Bezier curves for robotic path planning, the path curvature has to be taken into account. The calculation of the curvature of a Bezier curve is as follows:

$$\kappa(t) = \frac{P_x'(t)P_y''(t) - P_y'(t)P_x''(t)}{[P_x'(t)^2 + [P_y'(t)^2]^{3/2}} \tag{3.3}$$

where $P_x(t)$ and $P_y(t)$ represents x and y coordinates of the Bezier curve along the path parameter t with $0 \leq t \leq 1$ and $\bullet'$ denotes the derivative with respect to t. As special cases, the curvatures of the start and end positions of a Bezier curve are

$$\kappa(0) = \frac{2\,|(P_1 - P_0) \times (P_2 - P_1)|}{3\,|P_1 - P_0|^3} \tag{3.4}$$

and

$$\kappa(1) = \frac{2\,|(P_{n-1} - P_{n-2})x(P_n - P_{n-1})|}{3\,|P_n - P_{n-1}|^3} \tag{3.5}$$

While the above computation allows for validating the curvature of a single Bezier curve, one more issue arises when constructing paths by concatenating Bezier curves. In order to obtain drivable paths, the curvature of Bezier curves at the connection points needs to be the same. To this end, we consider the following important property of Bezier curves. Consider two Bezier curves as shown in Figure 3.2. Then, it holds that the two Bezier curves have the same curvature at the connection point if the last three control points of the first Bezier curve and the first three control points of the second Bezier curve are collinear.

Figure 3.2: Concated Bezier curves with collinear control points.

3.3 Proposed Solution to Find Smooth Paths

In Chapter 2, the VB and the VV-ST-R path are explained. However, these paths may have sharp corners. Since differential-drive or car-like robots cannot follow these sharp corners in a realistic scenario, the obtained path must not have these kinds of sharp corners The main contribution of this thesis is adding smooth curves to these obtained paths instead of sharp corners. In this section, the main components of the proposed algorithm are explained using several representative path segments from the example map in the previous chapter.

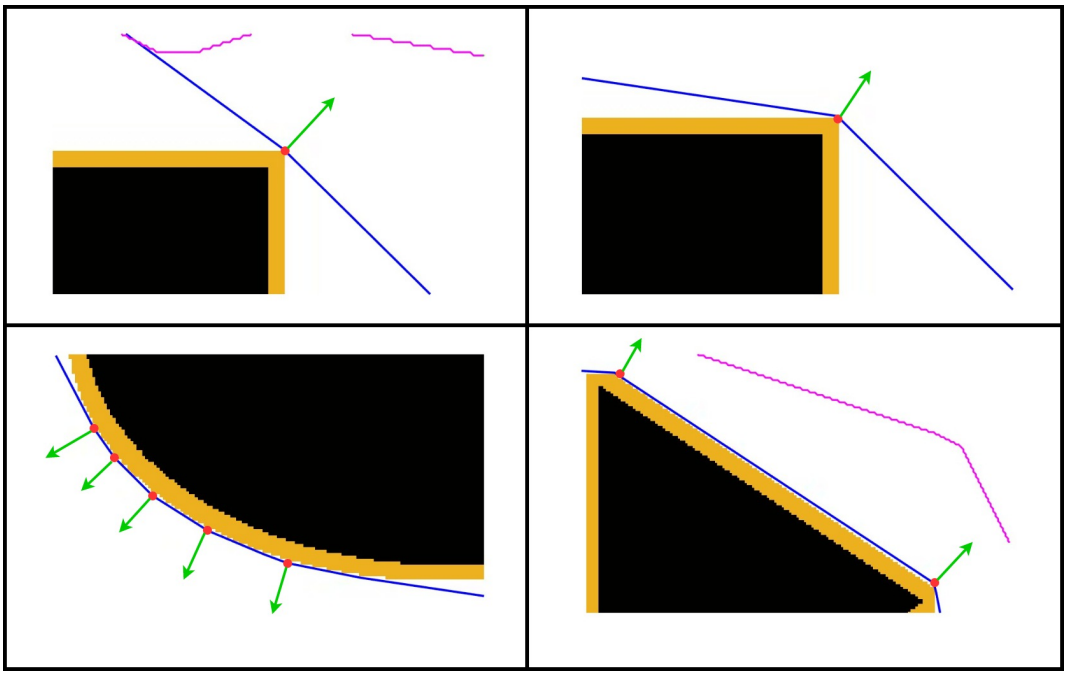

Figure 3.3: Example VV_ST_R path segments of the map in Figure 2.1.

First consider the left and right upper plot. Here, it is the case that a waypoint is placed at a corner of an obstacle. In the left plot, the path changes direction very little, whereas the path changes direction in the right plot. In both cases, it is desired to move the path away from the obstacle in the direction of the green arrow as indicated in the plot. Second, consider the lower right plot. Here, two waypoints are located at corners of a polygon-shape obstacle and the path directly follows the obstacle on a straight line. In this case, it is desired to move both waypoints away from the obstacle as indicated by the green arrows to create space for a smooth turn.

Finally, consider the lower left plot. Here, multiple waypoints are placed around a curved obstacle. In principle, it is desired to move the path away from the obstacle in direction of the green arrows. However, it can be observed that the waypoints are very close to each other, which makes it difficult to fit concatenated Bezier curves through the waypoints. Alternatively, it is possible to merge close waypoints and move the resulting merged waypoints away from the obstacle. This procedure is illustrated in Figure 3.5.

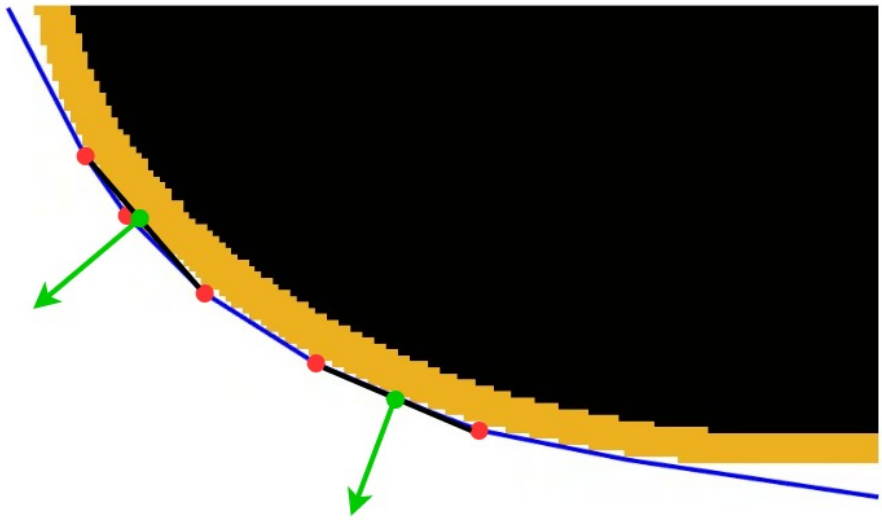

Figure 3.4: Example VV-ST-R Path

In the following sections, we first describe the procedures for merging and moving waypoints before explaining the overall algorithm for computing smooth paths.

3.3.1 Merging Close Waypoints

As was illustrated before, the VV-ST-R Algorithm in Section 2.2 performs turns by creating multiple points at points close to the obstacles. However, these corners are combined with straight lines, making it difficult to follow. In this thesis, we introduce a distance threshold Δ_{Dist}. If the distance between two consecutive waypoints is less than Δ_{Dist} (that is, waypoints are very close to each other), the mean value of these points are selected as new coordinates for the merged point. The pseudo code of

merging close waypoint is given in Step 1.

Step 1 Merge Close Points

1: Calculate distances between successive waypoints
2: **if** distance $< \Delta_{Dist}$ **then**
3: Generate new waypoint from mean value of close waypoints
4: Calculate Normal Vector ($\vec{N}$) pointing out from the obstacle
5: **end if**

Fig. 3.5 is given to illustrate the example of merging close waypoints and computing a suitable normal vector.

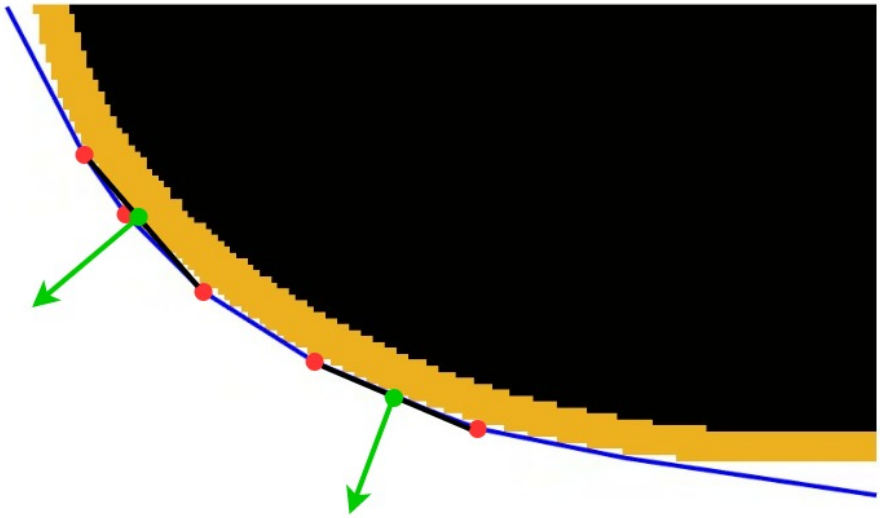

Figure 3.5: Merging waypoints together with the computation of a normal vector.

3.3.2 Move Waypoints Away from the Obstacle

New waypoints are obtained with the procedure described in the Step 1. These new waypoints will serve as the control points needed to create the Bezier curves and obtain smooth paths. These new points obtained may be close to obstacles or even merged control points may be in obstacle region. To prevent this situation, the points close to the obstacles have to be moved farther away from the obstacles. Hereby, we

use the normal vector $\vec{N}$ computed in Step 1 in the case of merged waypoints. For the remaining waypoints, we proceed as illustrated in Figure 3.6.

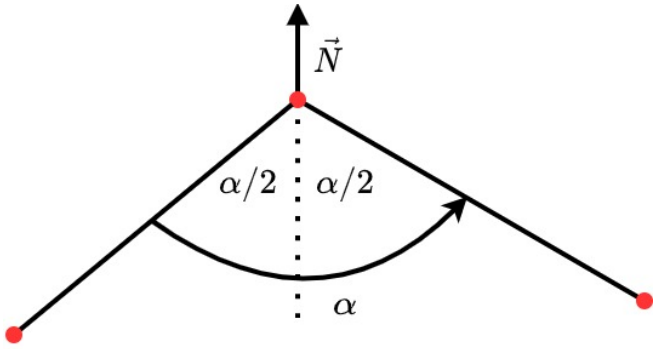

Figure 3.6: Normal vector computation for moving waypoints.

That is, first the angle α between two successive straight-line path segments is computed. Then, the normal vector $\vec{N}$ points in the direction of the corresponding bissectrice away from the obstacle.

Using the described procedures for computing the normal vector $\vec{N}$, the pseudo code for moving waypoints away from the obstacle is given in Step 2. Here, we introduce $m_{\min}$ and $m_{\max}$ as pre-defined parameters to indicate how much waypoints should be moved at least and at most, respectively. In addition, the parameter Δ_{Vb} is the distance between the current waypoint and the closest point on the VB.

Step 2 Move Points Away from Obstacle

1: Find closest point on VB for current waypoint: Δ_{VB}
2: **if** Waypoint is a merged waypoint **then**
3: Determine $\vec{N}$ as in Step 1
4: **else**
5: Use $\vec{N}$ ad described in Figure 3.6
6: **end if**
7: Move waypoint along $\vec{N}$ by $\max\{\min\{0.5 \cdot \Delta_{\mathrm{VB}}, m_{\max}\}, m_{\min}\}$

That is, the waypoint is moved between the obstacle and the VB as long as the minimum and maximum values $m_{\min}$ and $m_{\max}$ are not exceeded. The result of applying Step 2 is demonstrated for different values of Δ_{Dist} and $m_{\max}$ in Figure 3.7 and 3.8. Specifically, more points are merged and the waypoints are farther away from the

obstacles in Figure 3.8 since Δ_{Dist} and $m_{\max}$ are chosen larger.

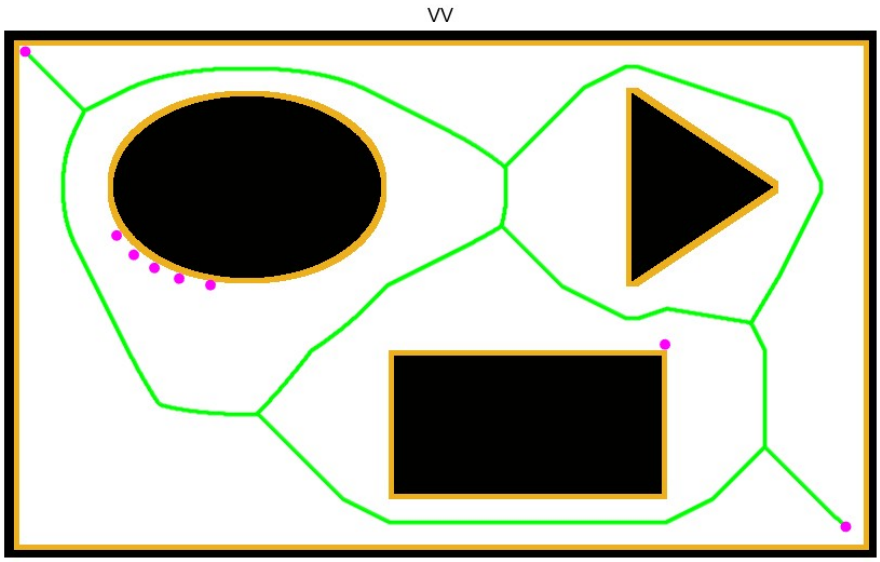

Figure 3.7: Moved waypoints in the example map.

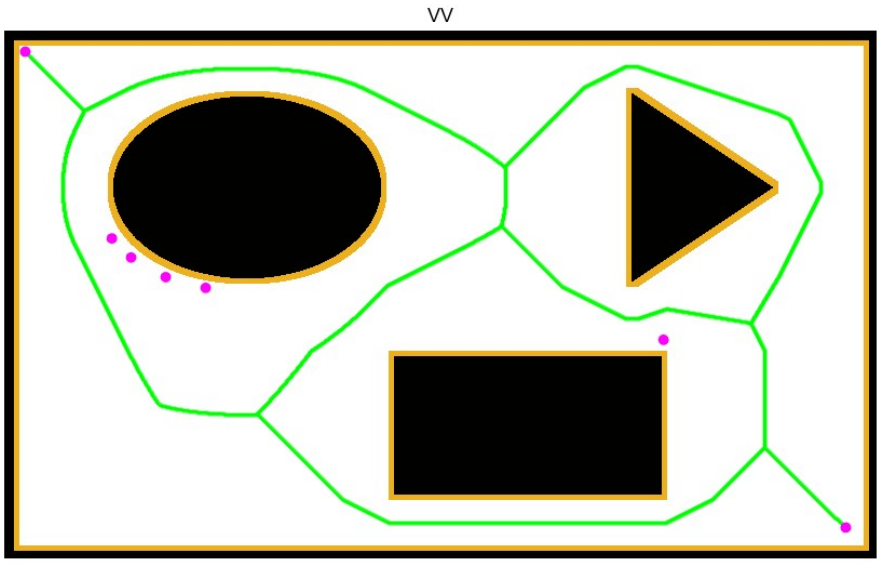

Figure 3.8: Moved waypoints in the example map for a larger value of Δ_{Dist} and $m_{\max}$.

3.3.3 Generate Helper Control Points for Bezier Curve

In this thesis, we propose to directly use the moved waypoints after applying the VV-ST-R algorithm and the procedure in Section 3.3.2 as control points for Bezier curves. Specifically, any two successive waypoints serve as the start and end point of a Bezier curve. In order to ensure a continuous curvature of concatenated Bezier curves at these start and end points, we apply the strategy described in Section 3.2. That is, for both start and end point, we insert two additional collinear "helper" control points. Hereby, we proceed as illustrated in Figure 3.9.

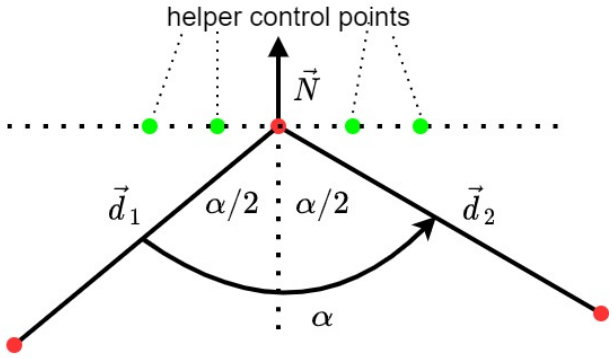

Figure 3.9: Helper control points

That is, we use the previously computed information of the moved waypoint and the associated normal vector $\vec{N}$ and determine the direction in which the helper waypoints are inserted as orthogonal to $\vec{N}$. Accordingly, a Bezier curve will enter any waypoint (that is used as control points) in the same direction as the concatenated Bezier curve leaves the waypoint.

In summary, we use Bezier curves of order six as a smooth curve between successive moved waypoints. In order to ensure a continuous curvature of the overall curve, helper control points are inserted at the start and end point of each Bezier curve. The first two helper control points are created near to the start point of Bezier curve. Step 3 explains the creation procedure of these two helper control points. We consider the W_i is the current waypoint and W_{i+1} is the next waypoint. Furthermore, $\vec{d_1} = W_{i+1} - W_i$ is the direction vector from the current control point to the next control point.

Step 3 Generate Two Helper Points Near First Control Point

1: Define $\vec{d_1} = W_{i+1} - W_i$
2: **if** W_i is the start position of the path **then**
3: Define $\vec{h_1}$ as a unit vector pointing along the orientation of the robot
4: **else**
5: Determine $\vec{h_1}$ as the direction of the last helper control points to the current Bezier curve segment's start point
6: **end if**
7: Create first helper control point as $H_{i,1} = W_i + \gamma \cdot \vec{h_1} \cdot ||\vec{d_1}||$
8: Create second helper control point $H_{i,2} = W_i + 2 \cdot \gamma \cdot \vec{h_1} \cdot ||\vec{d_1}||$

That is, the helper control points are inserted along a pre-defined direction with a distance that depends on the distance between the current waypoint W_i and the next waypoint W_{i+1} (which are used as control points). Here, the coefficient γ can be chosen to adjust the curvature of the Bezier curve around W_i as will be explored later in the thesis. Examples for the insertion of the helper control points are given in Figure 3.10.

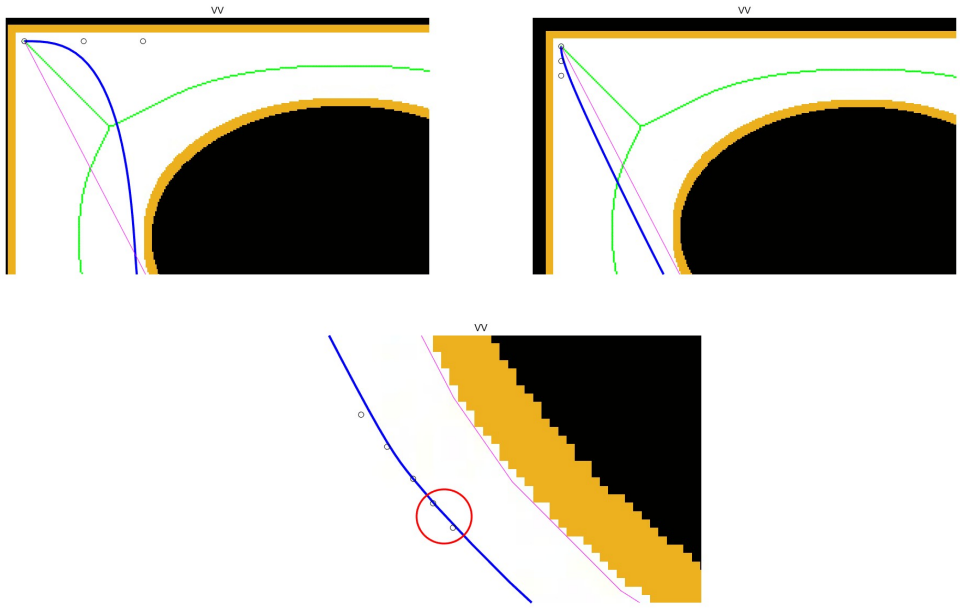

Figure 3.10: Helper control point examples.

It can be seen in the upper plots that the direction of the helper control points at the

start position is determined by the orientation of the robot, whereas the direction of the helper points at intermediate waypoints are dertermined by the direction of the previous helper control points in order to satisfy collinearity. Next, the insertion of the other two helper control points at the end point of the Bezier curve has to be discussed. Step 4 explains the creation procedure of these two helper control points. To this end, we recall the illustration in Figure 3.9. The current waypoint is denoted as W_i and the previous waypoint is W_{i-1}.

Step 4 Generate Two Helper Points Near Last Control Point

1: Calculate $\vec{d_2} = W_i - W_{i-1}$
2: **if** W_i is the end position of the path **then**
3: $\quad \vec{h_2} = \vec{d_2}/||\vec{d_2}||$
4: **else**
5: $\quad$ Calculate $\vec{h_2}$ as a unit vector that is orthogonal to the normal vector $\vec{N}$
6: **end if**
7: Create fourth control point near by the end point of current Bezier curve segment: $H_4 = W_i - 2 \cdot \gamma \cdot \vec{h_2} \cdot ||\vec{d_2}||$
8: Create fifth control point near by the end point of current Bezier curve segment: $H_5 = W_i - \gamma \cdot \vec{h_2} \cdot ||\vec{d_2}||$

In the Fig. 3.11, the creation of the helper points is shown. Since the control points around the start and end point of each Bezier curve are collinear, it makes it possible to have the same curvature of connected Bezier curve segments.

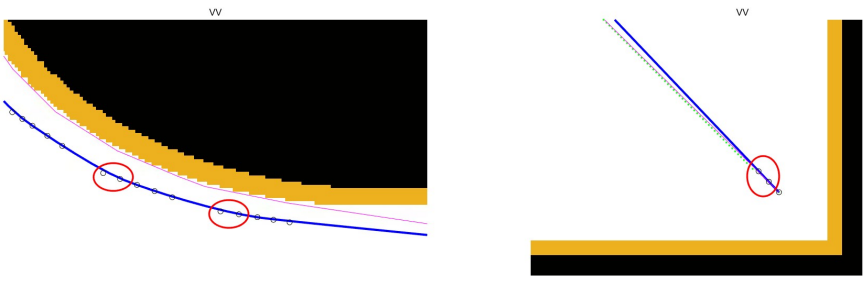

Figure 3.11: Generated helper control points at end point

3.3.4 Insert New Control Points for Bezier Curve

One of the main purposes of path planning algorithms is to find an obstacle-free path. During the creation of the Bezier curve, the algorithm needs to make sure that no obstacle is hit. To this end, the points on the Bezier curves are compared with the obstacle region. If any of these points hit an obstacle, the following procedure generates new waypoints in the obstacle-free space close to this point. By doing this, hitting an obstacle is avoided. Step 5 explains the creation procedure of inserting new waypoint in case any obstacle is hit. C_s is the start point, C_e is the endpoint of the current Bezier curve segment and C_o is the point hitting the obstacle. $\vec{d}_{se} = C_e - C_s$ is a direction vector from the start control point to the end control point of the Bezier curve segment. $\vec{d}_{obs} = C_o - C_s$ is a direction vector from the start control point of the Bezier curve segment to the point in the obstacle region.

Step 5 Insert New Waypoint Away From Obstacle

1: **if** Bezier curve point in obstacle region **then**
2: Calculate $\vec{d}_{se} = C_e - C_s$ from control point locations
3: Calculate $\vec{d}_{obs} = C_o - C_s$ from obstacle point and control points
4: Generate new control point as $C_s + \vec{d}_{se}^{T} \cdot \vec{d}_o$.
5: **end if**

That is, the new point is placed on the connection line between C_s and C_e, which is guaranteed to be in obstacle-free space.

3.3.5 Creation of Bezier Curves

Having explained the main components of our smooth path planning algorithm based on Bezier curves and the waypoints computed by the VV-ST-R algorithm, we can now summarize the steps in Algorithm 1. The algorithm first merges the close

Algorithm 1 Proposed Path Planning Algorithm

```
1: mergeClosePoints();
2: movePointsAwayFromCorner();
3: for iteration=2,...,Length Of Waypoints-1 do
4:     generateTwoHelperPointsNearFirstControlPoint();
5:     generateTwoHelperPointsNearLastControlPoint();
6:     createBezierCurveOfOrder6FromConsecutiveWaypoints();
7:     if Path hits obstacle then
8:         insertNewWaypointAwayFromObstacle();
9:         i := i − 1
10:    end if
11: end for
```

waypoints from the VV-ST-R algorithm as explained in Section 3.3.1. Then, the resulting waypoints are moved away from the obstacle borders or corners as described in Section 3.3.2. The helper control points at the start and end of each Bezier curve segment are then added as pointed out in Section 3.3.3. Finally, a Bezier curve is generated from the computed points and it is checked if the curve intersects with the obstacle region. If yes, the method for adding waypoints as introduced in Section 3.3.4 is applied.

Until now, only figures that illustrate the different steps of the algorithm were shown. Fig. 3.12 displays the path obtained as a result of the application of the full path planning algorithms. In this figure green lines represent the VB, the magenta line represents the VV-ST-R path, green and red filled points are the start and end points, black filled points are the helper control points and the blue line represents the generated smooth path.

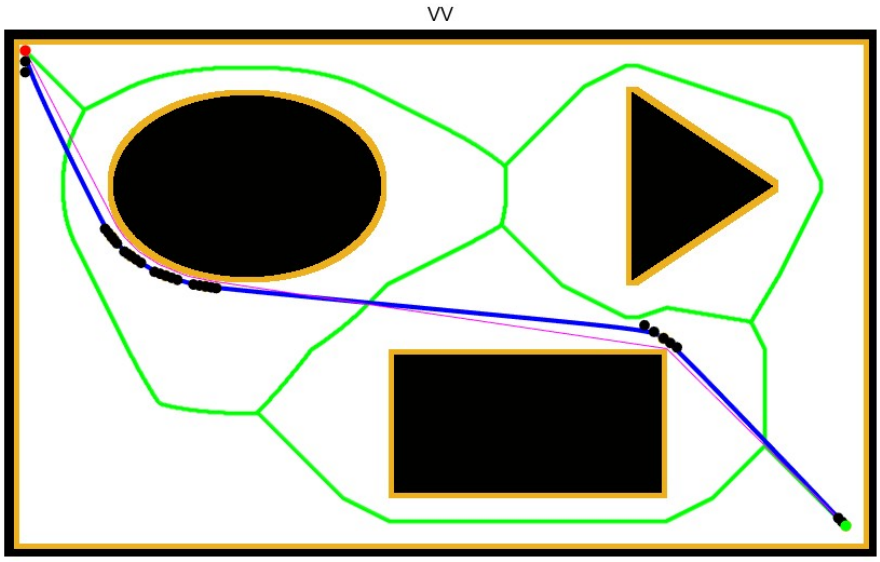

Figure 3.12: Generated Smooth Path with Bezier Curve Segments

3.3.6 Handling Curvature Constraints

The proposed algorithm in Algorithm 1, generates a smooth path by using Bezier curve segments. However, a curvature constraint value is not used in this algorithm. To handle a curvature constraint, the following Step 6 is added at the end of the Bezier curve creation.

Step 6 Handling Curvature Constraint

1: Introduce maximum curvature κ
2: **if** Bezier curve's curvature $> \kappa$ **then**
3: Increase the value of γ in Step 3 and 4 and re-compute the Bezier curve segment
4: **end if**

The results of the proposed Algorithm 1 in different maps, the effect of the curvature constraint value will be given in the next chapter.

CHAPTER 4

EXPERIMENTAL RESULTS

In this chapter, the differences between the proposed algorithm and the original algorithms will be shown. The comparison parameters are the path length, minimum distance to the obstacle and computation time. MATLAB is used for all calculations and figures in the experimental results section. All experiments run under the same condition with MATLAB on a laptop with Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz processor and 16GB Ram.

4.1 Selected Maps

There is a large variety of different maps in this thesis to try the proposed algorithm as shown in Figure 4.1. These maps are chosen to have different features to observe the performance of the proposed algorithm for maps with different characteristics. Map 1, 2, 5, 6, 7, 8, 9, 16, 21, 24, 25, and 26 have multiple possible paths between the same start and goal positions. Other maps have only one path between the start and goal position. Also, there are more different characteristics apart from the possible path numbers. Some maps (i.e., Map 1, 7, 23, 25, . . .) are sparse maps such that robots can move easily. On the other hand, some maps (i.e., Map 10, 12, 16, 21,. . .) have narrow passages between the obstacles such that the maneuverability of the robot is limited. These narrow passages affect the curvature of the path. In the following sections, different experiments on these maps are explained. The proposed algorithm's effect is examined on narrow passages and wide free spaces.

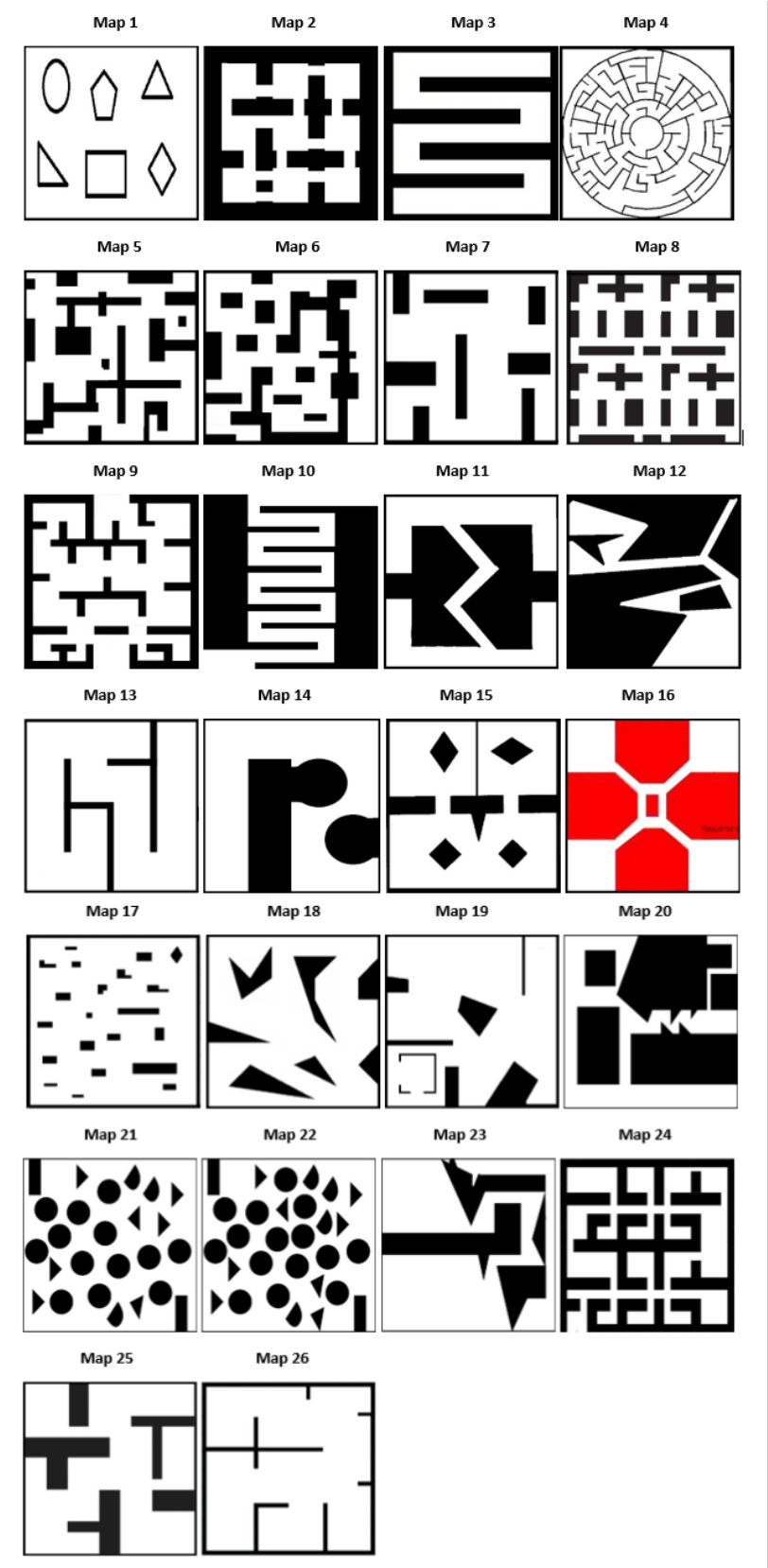

Figure 4.1: Used Maps List

4.2 Application of the Proposed Algorithm without Curvature Constraint

The main purpose of the proposed algorithm is to find a smooth path between start and goal positions. For this purpose, the proposed algorithm uses pre-calculated waypoints from the VV-ST-R algorithm. The VV-ST-R also uses the VB as its basis. Since the path along the VB has a long path length, VV-ST-R focuses to find a shorter path. However, both of the algorithms find paths that are composed of straight lines. Therefore, nonholonomic robots such as differential-drive robots or car-like robots cannot follow these paths. The proposed algorithm aims to use the safe area between the VB and the VV-ST-R paths. Therefore, it is expected that the proposed algorithm can find a safe and drivable path for nonholonomic robots. For the result comparison, the minimum distance to the obstacle, and total path length are selected. Also, the computation time for different maps is compared.

4.2.1 Comparison with Path Length

The first comparison parameter is the path length. The VB aims that the safest path between start and goal positions. Thus, it places its waypoints away from the obstacle, and the resulting path becomes longer. VV-ST-R aims that the shortest path between the start and goal position. Thus, it places its waypoints near the obstacle and its path is shorter than the VB path. Since the proposed algorithm uses a safe region between these two algorithms to place Bezier curve control points, the resulting length is expected in the middle of these two algorithms.

In Fig. 4.3 and Fig. 4.2, there are sparse maps. Therefore, the VB path's length is much larger than the other two algorithms. The proposed algorithm's path length is between the VB path and VV-ST-R path but closer to the VV-ST-R.

In Fig. 4.6, Fig. 4.5 and Fig. 4.4, there are sharp corners and tight passages between the obstacles. Since there is limited free space, the path length difference between VB and VV-ST-R is smaller than the other maps. In this kind of maps, the proposed algorithm's path length is still closer to the VV-ST-R path. The difference between the proposed algorithm's path length and VV-ST-R's path length is because the proposed algorithm changes the locations of control points more distant from the obstacles.

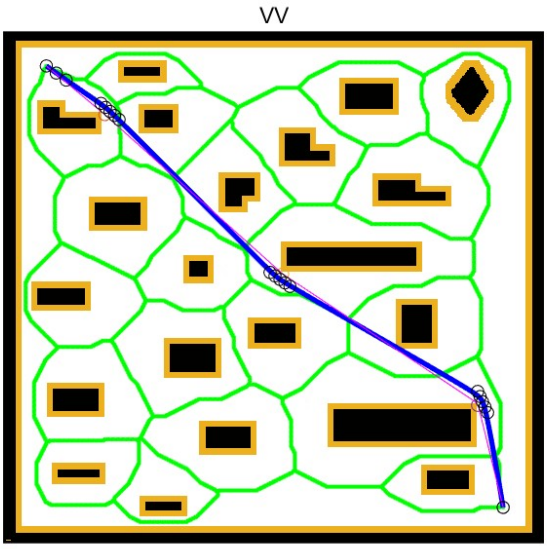

Figure 4.2: Obtained Smooth Path for Map 17

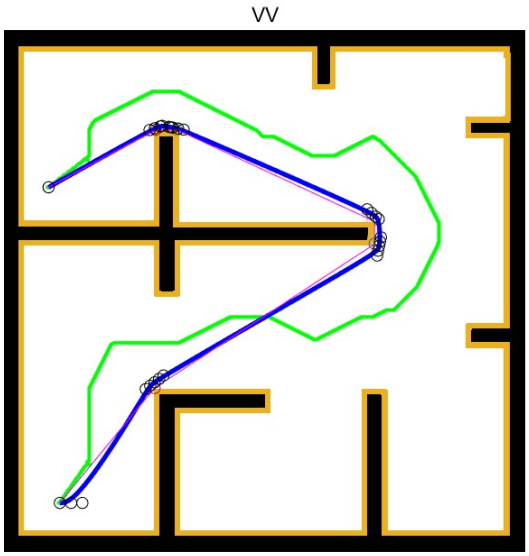

Figure 4.3: Obtained Smooth Path for Map 26

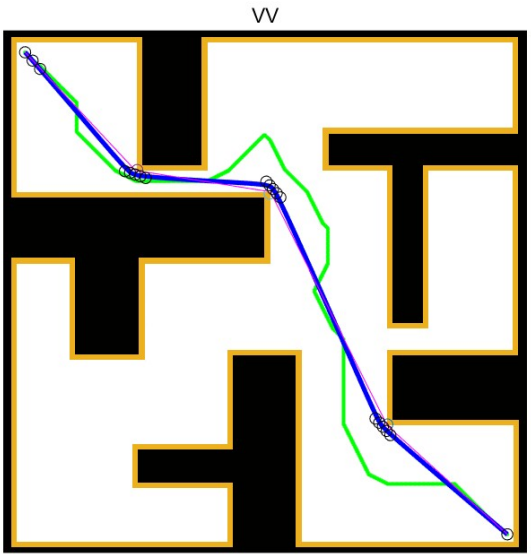

Figure 4.4: Obtained Smooth Path for Map 25

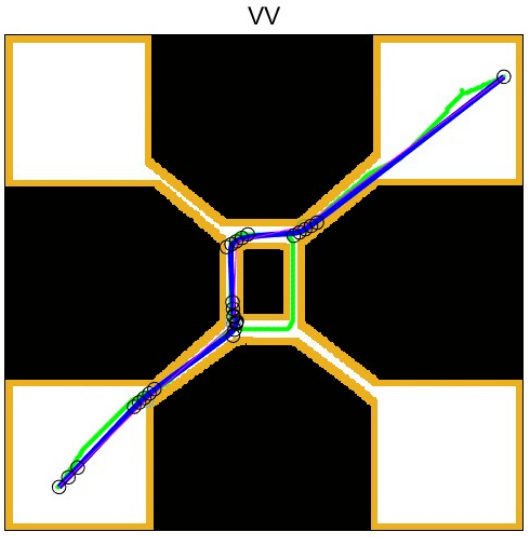

Figure 4.5: Obtained Smooth Path for Map 12

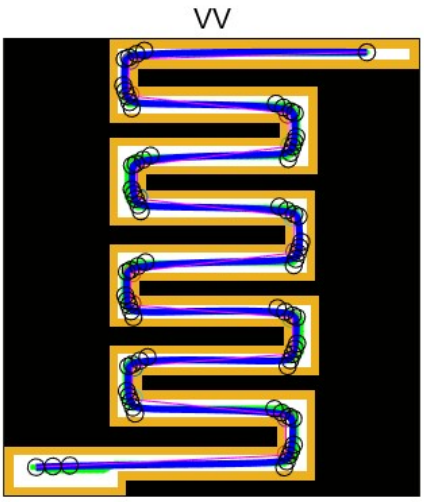

Figure 4.6: Obtained Smooth Path for Map 10

The path length is affected by the sharpness of the corners and the type of passages between obstacles. It is known that VB places its waypoint as far as possible from the nearest obstacle. Also, VB waypoints are equal distances away from the adjacent obstacles. On the other hand, VV-ST-R algorithm places its waypoint as much as closer to the obstacle to obtain the shortest path. If obstacles are so close to each other both of the algorithms generate their waypoints almost similar locations between the obstacles. Then, the path length is affected by this situation. In conclusion for this comparison, the proposed algorithm can find path length in the middle of VB and VV-ST-R paths. In sparse maps, VV-ST-R's path length is much shorter than the VB path. The proposed algorithm finds a path whose length of the path is closer to the VV-ST-R path. In maps with narrow passages, the proposed algorithm can find similar path lengths with both the VB and VV-ST-R algorithms because free areas are limited and waypoints generated almost similar locations. For all the maps, different from VV-ST-R and VB paths, the proposed algorithm's solution includes smooth curves not, sharp corners. In Table 4.1, the path length comparison is shown for three different algorithms.

Path Length (px)			
Map	**Voronoi Diagram**	**VV-ST-R**	**Bezier Curve Implementation**
Map 1	655.0	504.8	511.6
Map 2	549.9	428.5	441.8
Map 3	1472.4	1253.6	1310.2
Map 4	1064.9	910.5	923.7
Map 5	943.6	750.4	770.5
Map 6	827.1	686.3	703.7
Map 7	702.4	552.9	573.9
Map 8	648.6	503.7	524.5
Map 9	635.6	512.7	519.1
Map 10	1224.7	1100	1160.3
Map 11	1210.8	989.4	1020.3
Map 12	794.6	681.3	696.8
Map 13	1797.4	1276.7	1310.4
Map 14	1246.5	946.2	970.8
Map 15	956.8	677.1	694.4
Map 16	503.1	472.8	480.1
Map 17	666.9	538.5	544.1
Map 18	809.1	599.6	607.5
Map 19	705.3	575.9	592.3
Map 20	1411.1	1262.5	1299.7
Map 21	1077.5	864.4	868.4
Map 22	1094	873.1	886.1
Map 23	2042.6	1734.6	1761.2
Map 24	757.6	698.3	719.1
Map 25	790.2	642.7	652.1
Map 26	1004.7	704.3	717.3

Table 4.1: Path Length Results Comparison

4.2.2 Comparison with Minimum Distance to Obstacle

The second comparison parameter is the minimum distance to the obstacle. The VB path aims at the safest path between the start and goal positions. VV-ST-R aims at the

shortest path between the start and goal position. Therefore, the minimum distance is smaller for the VV-ST-R algorithm. However, both of the algorithms cannot produce a drivable path. Since the proposed algorithm generates smooth curves instead of straight lines between the VB path and the VV-ST-R path, the control points of the Bezier curve are placed at a safe distance between the VB path and the VV-ST-R path. Therefore, the minimum distance parameter of the proposed algorithm is between the VB and VV-ST-R algorithms.

In Fig. 4.7 and Fig. 4.8, there are sparse maps. The pink line represents the obtained VV-ST-R path and the green line represents the VB path. The resulting smooth path is shown with blue lines. Although the free spaces are wide, the VB aims to find the safest path by giving the distance to the nearest obstacle, its minimum distance to the obstacle parameter is larger than the other algorithms. Since the proposed algorithm tries to find smooth paths it creates control points away from the obstacles. Thus, its minimum distance to the obstacle parameter is between the VB and VV-ST-R algorithms but much closer to the VV-ST-R path.

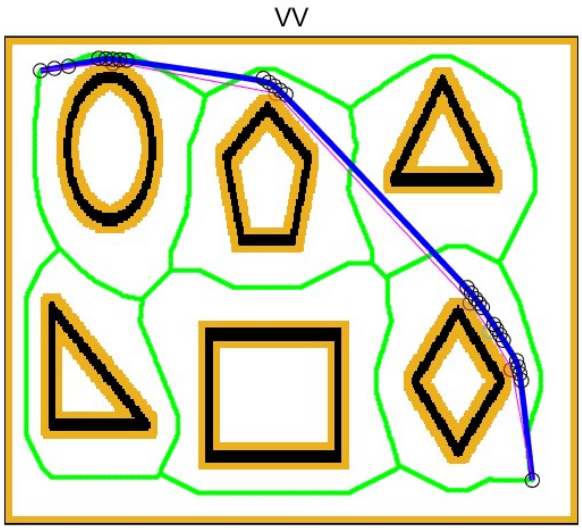

Figure 4.7: Obtained Smooth Path for Map 1

On the other hand, in Fig. 4.9 and Fig. 4.10 there are narrow passages between obstacles. Therefore, the minimum distance to the obstacle parameter is almost the same

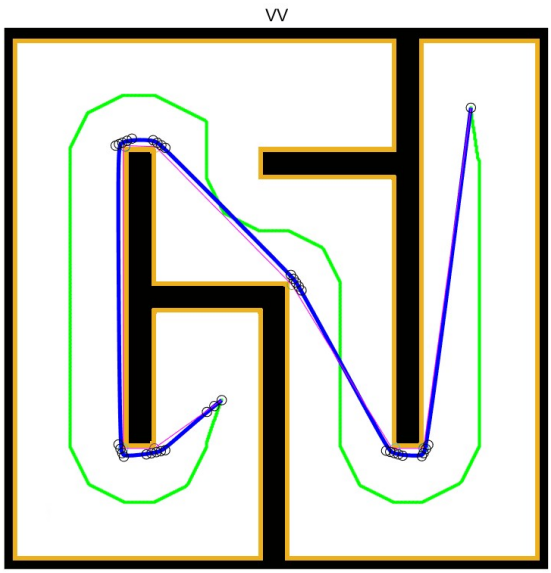

Figure 4.8: Obtained Smooth Path for Map 13

for all three algorithms. Nevertheless, the proposed algorithm finds a smooth path between the start and goal position.

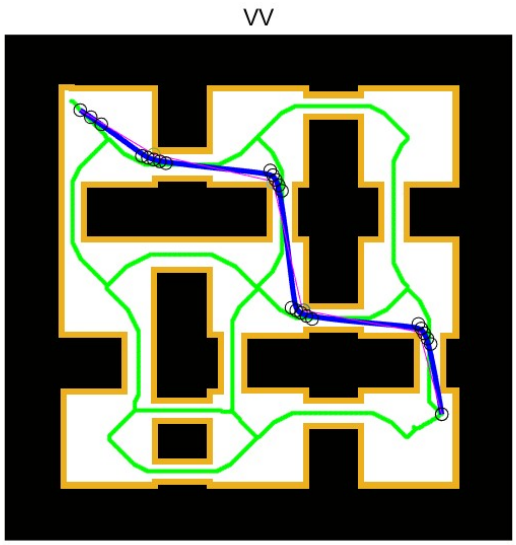

Figure 4.9: Obtained Smooth Path for Map 2

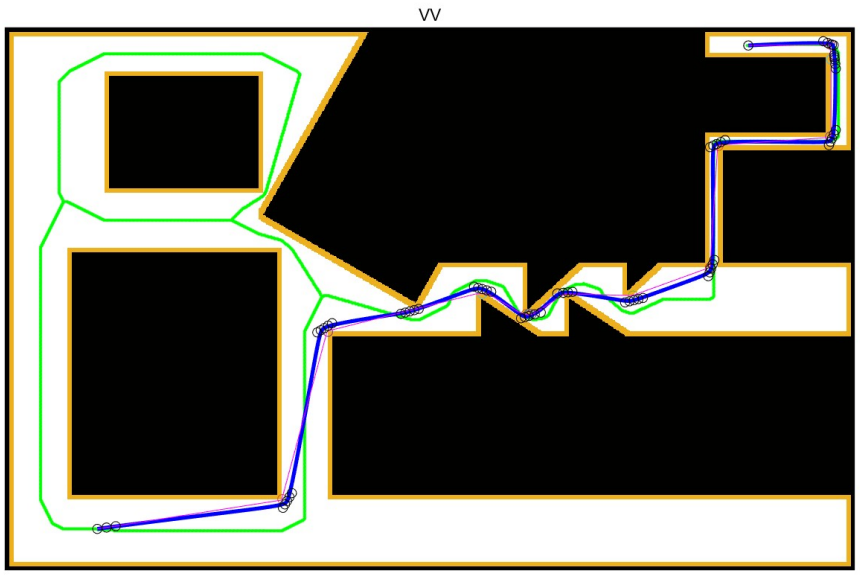

Figure 4.10: Obtained Smooth Path for Map 20

The second comparison parameter is the minimum distance to the obstacle. The result of this comparison is similar to the path length comparison. Also, both comparison parameters are affected by each other. Since the VB algorithm places its waypoints as much as away from the obstacle the minimum distance to the obstacles becomes large and the path length will increase. On the other hand, the VV-ST-R algorithm places its waypoints near the obstacle, and the path length will decrease. From the nature of both VB and VV-ST-R algorithms, these two comparison parameters give similar results to each other. In conclusion for this comparison, the proposed algorithm can find the minimum distance to the obstacle in the middle of VB and VV-ST-R paths if there are no narrow passages along the map. However, the proposed algorithm's result is closer to the VV-ST-R algorithm. In narrow passages, all three algorithms find similar distances to the obstacle due to the limited free space. In Table 4.2, the minimum distances to the obstacle are shown for three different algorithms. As shown in that table, the minimum distance value for the proposed algorithm is between the other two algorithms' values for all of the maps. However, it is much closer to the VV-ST-R path's results.

Minimum Distance to Obstacle (px)			
Map	**Voronoi Diagram**	**VV-ST-R**	**Bezier Curve Implementation**
Map 1	12	6	8.7
Map 2	12	6.5	7.6
Map 3	14	5.7	7.7
Map 4	14.3	6.1	7.6
Map 5	16	6.1	8.1
Map 6	9	6.1	7.7
Map 7	20	6.0	10.8
Map 8	13	6.5	9.8
Map 9	13	6	7.1
Map 10	8	6	7.4
Map 11	16.2	6	8.1
Map 12	10	5.8	7.6
Map 13	45	5.9	7.7
Map 14	20.1	5.8	10.2
Map 15	17	6.1	9.3
Map 16	8.2	6.1	6.8
Map 17	20	7.1	6.1
Map 18	15	6.1	7.4
Map 19	12	6.2	9.1
Map 20	8.9	5.8	6.8
Map 21	16	7.4	8.9
Map 22	6	6.1	6.2
Map 23	11	6.1	7.6
Map 24	8	6	6.4
Map 25	11	6.3	10.3
Map 26	27	6.1	7.8

Table 4.2: Minimum Distance to Obstacle Results Comparison

4.2.3 Comparison with Computation Time

The third comparison parameter is computation time. The proposed algorithm uses previously computed VV-ST-R waypoints as control points of the Bezier curve seg-

ments. Therefore, the computation times of previous algorithms are not included in the comparison. Only the computation time of the proposed algorithm for different maps is compared.

In Fig. 4.11 and Fig. 4.12, the smooth path around the circular shapes is shown. Since more control points are calculated one after the other to pass circular obstacles, computation time is long in maps containing more control points like these maps.

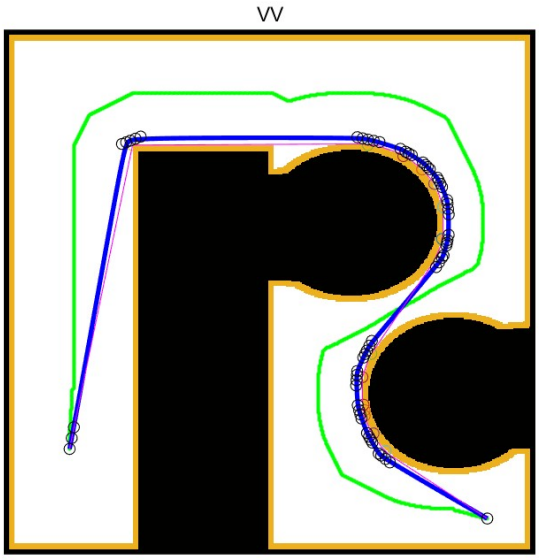

Figure 4.11: Obtained Smooth Path for Map 14

In Fig. 4.14 and Fig. 4.13, the smooth path between sharp obstacle ends is shown. Since these are sparse maps, there are not many control points produced for this map. Fewer control points are generated to give direction to the vehicle during turns. Therefore, computation time is less in this and similar maps.

In conclusion for this comparison, the proposed algorithm's computation time varies according to the number of control points on the road. If there is a path on the map that needs to be passed around detailed obstacles, the computation time is longer. If the distance between the obstacles on the map is large and there is no need for more control points to find the way, the computation time is shorter. In Table 4.3, the computation time is shown for different maps with the proposed algorithm.

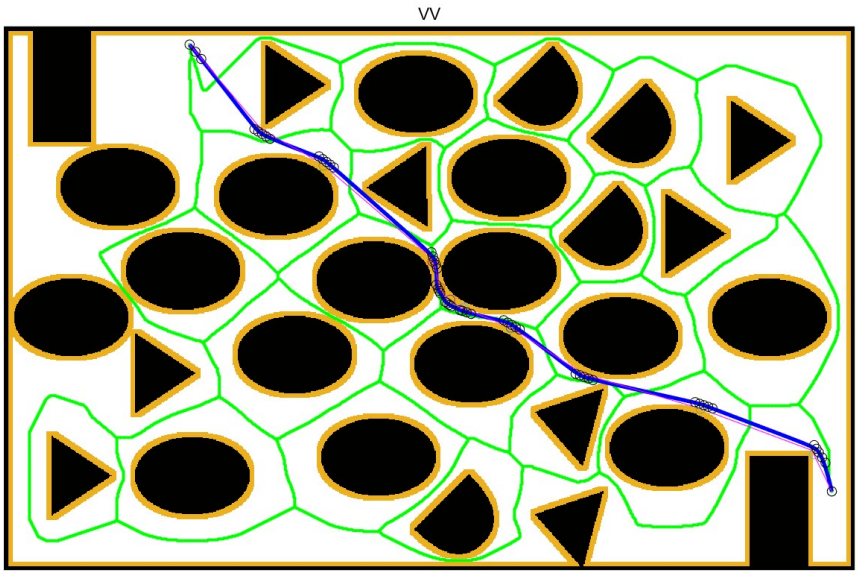

Figure 4.12: Obtained Smooth Path for Map 22

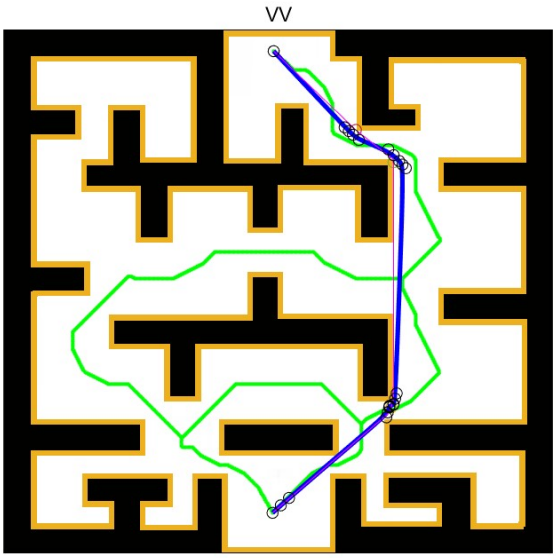

Figure 4.13: Obtained Smooth Path for Map 9

Computation Time (seconds)	
Map	**Bezier Curve Implementation**
Map 1	0.3373
Map 2	0.1188
Map 3	0.1584
Map 4	0.1149
Map 5	0.1293
Map 6	0.1413
Map 7	0.0997
Map 8	0.1132
Map 9	0.0703
Map 10	0.2419
Map 11	0.1966
Map 12	0.1458
Map 13	0.1122
Map 14	0.2885
Map 15	0.1085
Map 16	0.0743
Map 17	0.0601
Map 18	0.0556
Map 19	0.0869
Map 20	0.2548
Map 21	0.1355
Map 22	0.1544
Map 23	0.1078
Map 24	0.1502
Map 25	0.0710
Map 26	0.0919

Table 4.3: Computation Time Results Comparison between Different Maps

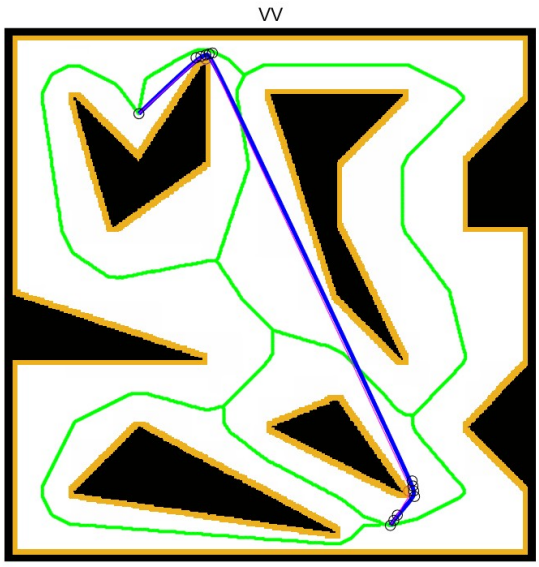

Figure 4.14: Obtained Smooth Path for Map 18

4.3 Evaluation of Curvature Constraint

In this section, the results for the extended algorithm in Section 3.3.6 with curvature constraint are shown. For the different curvature constraints, the comparisons are made according to the total path length and minimum distance to the obstacle. For these experiments, some example maps are selected. Since these maps have significant changes when curvature is changed, these maps are more suitable to show the difference between different curvature constraints. Map 3, Map 13, Map 18, and Map 24 are selected to show differences both visually and in parameter comparisons.

4.3.1 Comparison with Path Length

In this section, a path length comparison is given. The comparison is made with different curvature (κ) values. As shown in Table 4.4, path length increases when a smaller curvature constraint is set. The main idea for addressing the curvature constraint is changing control point locations to obtain more smooth curves. It is known that the control point's locations have a direct effect on the Bezier curve. To handle curvature constraints, the proposed algorithm places helper control points more far

away from each other. Thus, the resulting Bezier curve becomes more smooth. This causes an extension in the path length. In Table 4.4 maps have more changes than the other maps in the path length when the curvature constraint is smaller. In addition, these maps were chosen because it is easier to visually see the change in path in these maps.

Path Length (px)			
Map	No Constraint	$\kappa = 0.5$	$\kappa = 0.25$
Map 3	1310.3	1310.3	1314.1
Map 13	1310.4	1310.4	1313.4
Map 18	607.5	608.3	609.7
Map 24	719.1	719.2	720.4

Table 4.4: Path Length Comparison with Different Curvature Constraints

4.3.2 Comparison with Minimum Distance to Obstacle

In this section, the comparison of the minimum distance to the obstacle is given. The comparison is made with different curvature (κ) values. As shown in Table 4.5, the minimum distance to obstacle value for Map 3 and Map 24 decreased as the curvature value decreased. This has been observed because the algorithm brings the path closer to the far corners of the corridors to reach the desired curvature value. There were no serious changes for Map 13, and Map 18. In these maps, the obstacle-free area is wider and the curvature value is reached more easily.

In Fig. 4.15, and Fig. 4.16 different obtained paths for Map 3 are given with no curvature constraint, and κ=0.25 respectively. There are u-turns on this map. To reach the desired curvature value in u-turns, the algorithm has increased the distance between the control points. In this way, there have been changes in the turns on the map. In Fig. 4.17, the distance change between control points and the path's shape change is shown.

In Fig. 4.18, and Fig. 4.19 different obtained maps for Map 13 are given with no curvature constraint, and κ=0.25 respectively. On the left side of this map, there are

Minimum Distance to Obstacle (px)			
Map	**No Constraint**	**K = 0.5**	**K = 0.25**
Map 3	7.77	7.77	7.74
Map 13	7.81	7.77	7.77
Map 18	7.42	7.41	7.41
Map 24	6.46	6.46	6.01

Table 4.5: Minimum Distance to Obstacle Comparison with Different Curvature Constraints

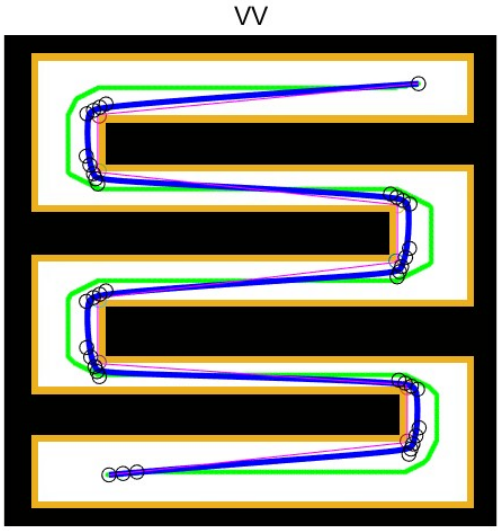

Figure 4.15: Obtained Smooth Path for Map 3 with No Curvature Constraint

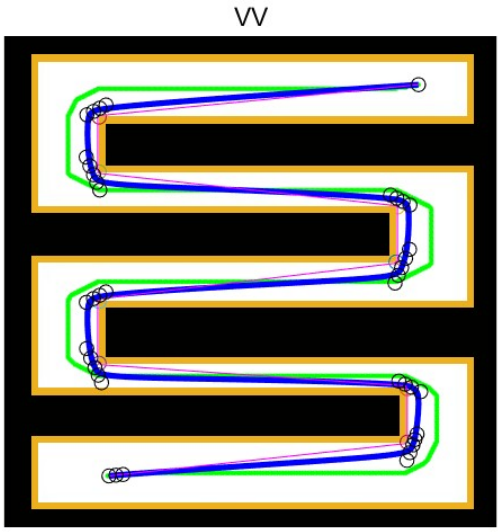

Figure 4.16: Obtained Smooth Path for Map 3 with κ=0.25

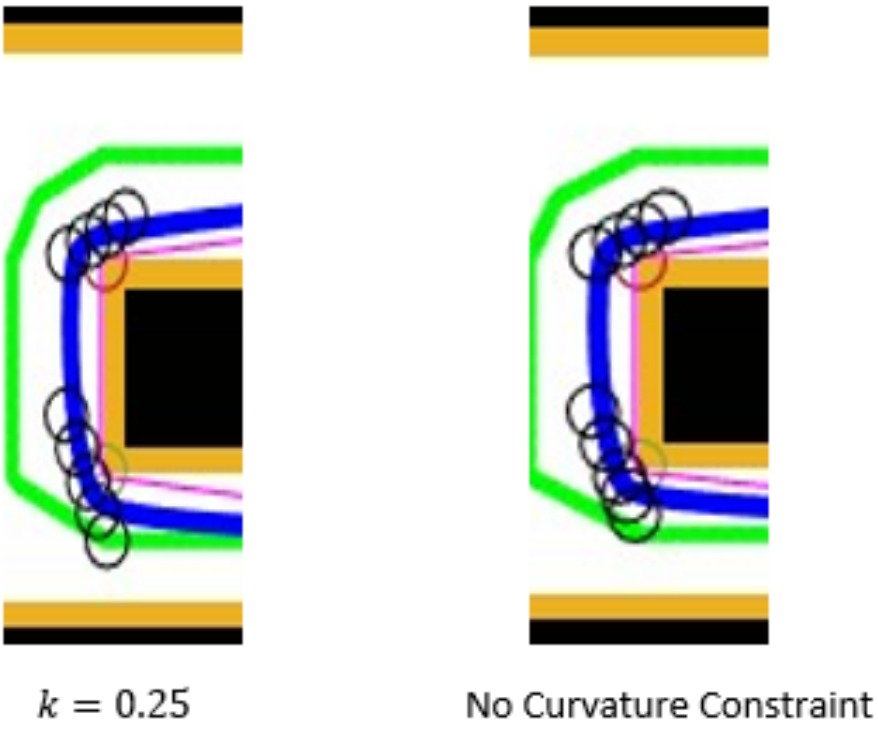

Figure 4.17: Close Up to Path Shape Difference for Map 3

steep turns before the path progresses in a straight line. To reach the desired curvature value, the algorithm has moved the control points away from each other. Since this is a sparse map, the curvature value is handled by extending the path toward the empty areas. In Fig. 4.20, the distance change between control points and the path's shape change is shown.

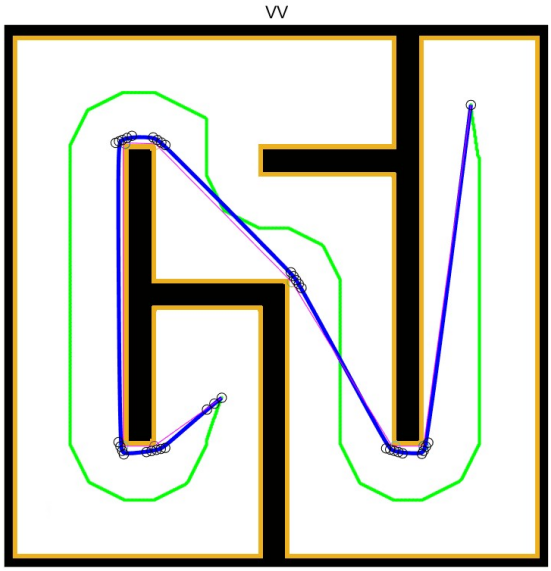

Figure 4.18: Obtained Smooth Path for Map 13 with No Curvature Constraint

In Fig. 4.21, and Fig. 4.22 different obtained maps for Map 18 are given with no curvature constraint, κ=0.5, and κ=0.25 respectively. In this map, there is a sharp turn on the pointed end of the obstacle at the top of the map. As the desired curvature value decreases, the algorithm further increases the distance between control points, and the desired curvature constraint is handled in this turn. In Fig. 4.23, the distance change between control points and the path's shape change is shown.

In Fig. 4.24, and Fig. 4.25 different obtained maps for Map 24 are given with no curvature constraint, κ=0.5, and κ=0.25 respectively. The distance between obstacles in this map is both narrow and includes sharp turns. In these sharp turns, the algorithm was able to find the desired curvature value without violating the safety constraint. In Fig. 4.26, the distance change between control points and the path's shape change is shown.

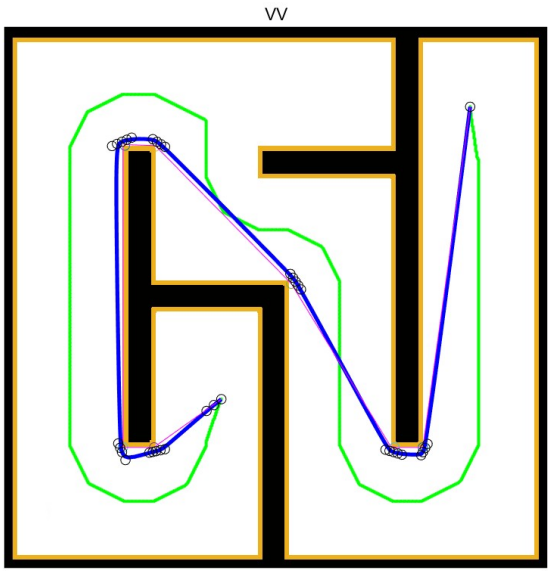

Figure 4.19: Obtained Smooth Path for Map 13 with κ=0.25

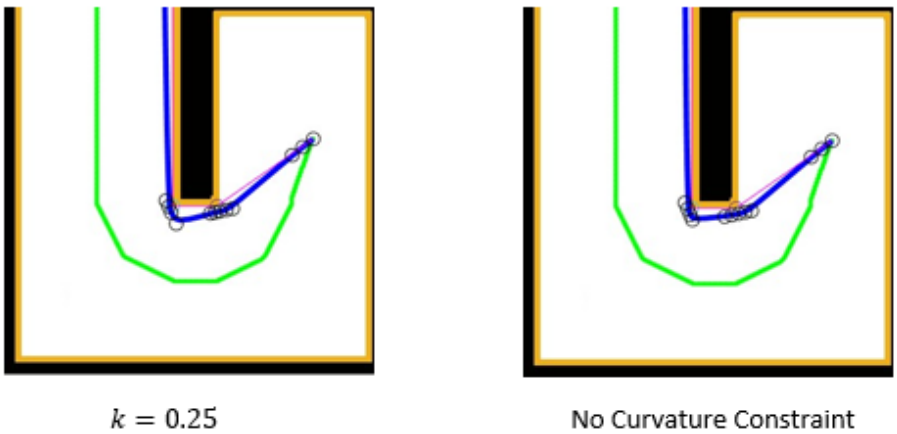

Figure 4.20: Close Up to Path Shape Difference for Map 13

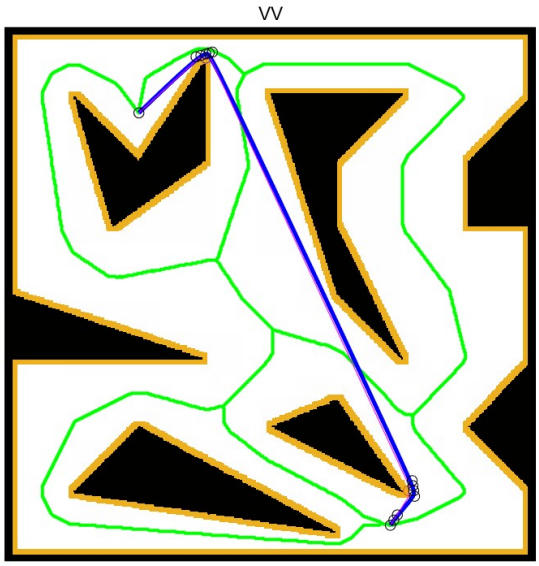

Figure 4.21: Obtained Smooth Path for Map 18 with No Curvature Constraint

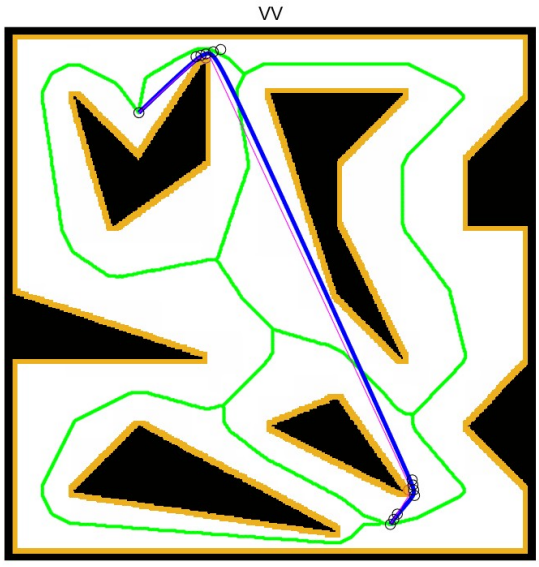

Figure 4.22: Obtained Smooth Path for Map 18 with κ=0.25

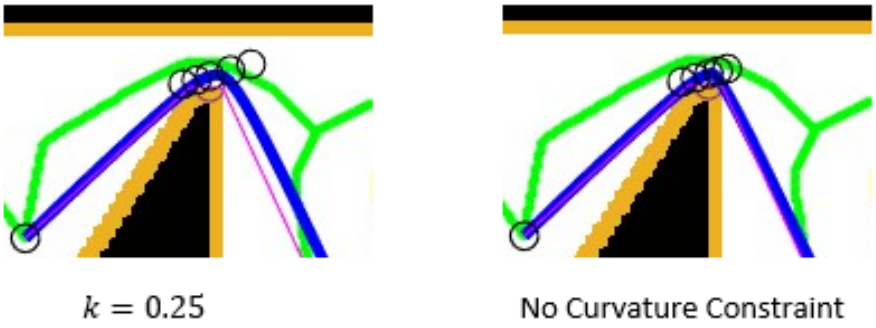

Figure 4.23: Close Up to Path Shape Difference for Map 18

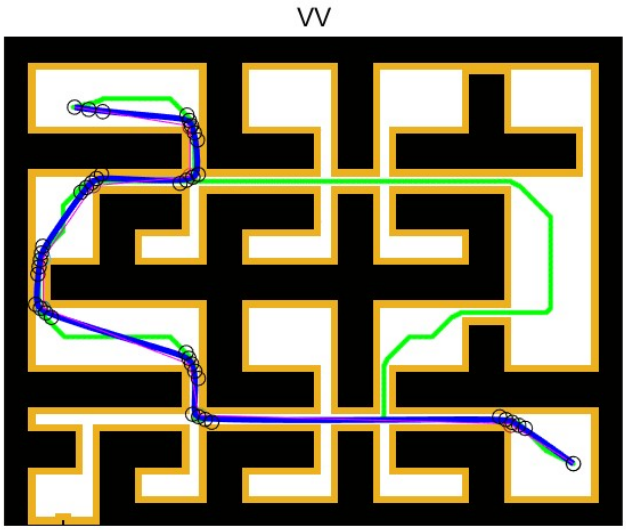

Figure 4.24: Obtained Smooth Path for Map 24 with No Curvature Constraint

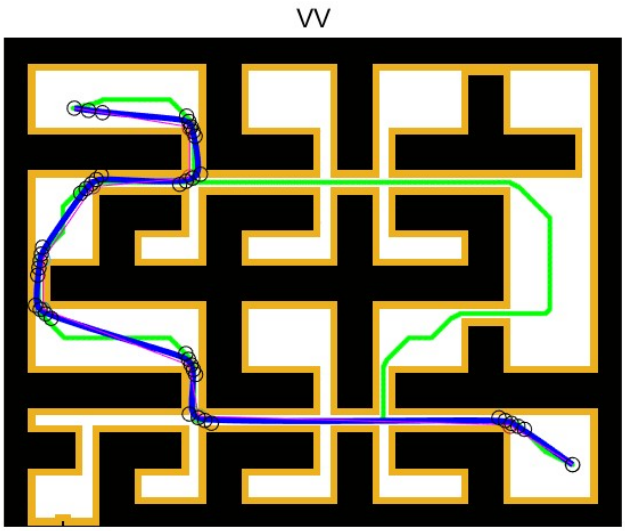

Figure 4.25: Obtained Smooth Path for Map 24 with κ=0.25

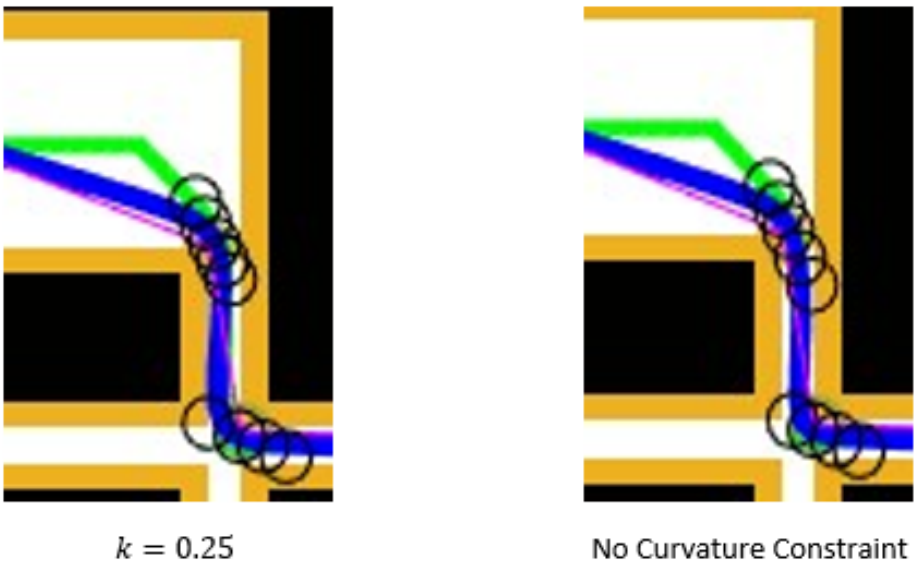

Figure 4.26: Close Up to Path Shape Difference for Map 24

4.4 Validation of the Quality of Obtained Paths

MATLAB Robot Simulator [29] was used to test the traceability and quality of the smooth paths obtained. In this tool Pure Pursuit path tracking controller is used. In path tracking tests, both differential drive robot kinematics and Ackermann steering kinematics were used. As a result of the trials with this tool, if the proposed algorithm produces smooth paths, the robots can follow the path. However, if smooth paths cannot be produced, an overshoot situation occurs, and the vehicle cannot follow the path. From [30], we can understand why smooth paths are needed for traceability. The pure pursuit algorithm identifies a target point a certain distance from the vehicle's current position. It tries to reach this determined point by drawing a circular arc.

The target point determined during the autonomous movement continuously shifts according to the robot's position. Thus, it creates a smooth and traceable path. The look-ahead distance determines the target point determined here. The projected point of the specified point is calculated. A triangle is created with the target point, the projected point, and the vehicle's instantaneous position. This way, the angle value required to reach the target is calculated. Thus, the steering angle value is calculated, and the vehicle is provided to go to the determined point on a curve. Since the vehicle cannot maneuver on a point, it needs some distance to maneuver. However, this distance will be considerably increased by sharp turns on the road. Therefore, if there is a sharp turn on the path, the algorithm will create a curve to follow it. This will cause it to overshoot from the planned path. For this reason, the vehicle must have a smooth curve to follow.

Ackermann Steering		Differential Drive		
Wheel Base	Lookahead Dist.	Track Width	Wheel Radius	Lookahead Dist.
0.5 m	2 m	0.5 m	0.1 m	1 m

Table 4.6: Used Parameters for Ackermann Steering and Differential Drive

In the validation the quality of obtained path, both car-like robot with Ackermann steering and differential drive robot are used. Compared to the straight-line maps found with the VV-ST-R algorithm, the smooth path created with the Bezier curves

found by the proposed algorithm could be followed better by the robots. It has been observed that the vehicle can follow the path at different curvatures κ=0.5 and κ=0.25. In Table 4.6, used parameters are given that used during the experiments.

In Table 4.7 and Table 4.8, tracking errors as a mean and maximum distance to the generated smooth path for differential drive robot and Ackermann steering robot are shown. In these tables, tracking errors of three different curvature values which are κ=1, κ=0.50, and κ=0.25, are shown as maximum and mean. The less these values are, the better the path can be followed. Mean and maximum values of errors in following the path found by the proposed algorithm are shown in the following tables. For both differential drive and Ackermann steering robots, the robots can successfully follow the paths generated with different curvatures. The main purpose of this experiment is to show the relationship between the curvature value and the traceability of the path. As mentioned in Section 4.3, the effect of the curvature change varies according to the characteristic features of the map. This effect also showed itself in the tracking errors section. Tracking errors have also changed according to the changes in the path.

The biggest effect of the curvature value on following the path is shown in the Table 4.8 for Map 15. For the large curvatures, maximum tracking distance to the generated path is higher than κ=0.25. Although the changes due to curvature in other maps were not very large, the robots were able to successfully follow the smooth paths with robots for each curvature value.

Tracking Errors for Differential Drive Robot (meters)						
	K=1		**K=0.5**		**K=0.25**	
Map	Mean(m)	Max(m)	Mean(m)	Max(m)	Mean(m)	Max(m)
Map1	0.1705	0.4538	0.1705	0.4538	0.1705	0.4538
Map2	0.1710	0.4416	0.1710	0.4416	0.1710	0.4416
Map3	0.1753	0.4623	0.1753	0.4623	0.1775	0.4559
Map4	0.1725	0.4608	0.1725	0.4608	0.1725	0.4608
Map5	0.1661	0.4800	0.1661	0.4800	0.1660	0.4645
Map6	0.1755	0.4522	0.1755	0.4522	0.1748	0.4547
Map7	0.1722	0.4556	0.1722	0.4556	0.1722	0.4556
Map8	0.1814	0.5837	0.1814	0.5837	0.1814	0.5837
Map9	0.1671	0.5165	0.1668	0.5945	0.1668	0.5945
Map10	0.1795	0.5708	0.1795	0.5708	0.1838	0.5819
Map11	0.1711	0.4794	0.1711	0.4794	0.1711	0.4794
Map12	0.1660	0.4699	0.1660	0.4699	0.1845	0.7830
Map13	0.1818	0.4883	0.1818	0.4883	0.1785	0.4702
Map14	0.1598	0.4706	0.1598	0.4706	0.1588	0.4623
Map15	0.1693	0.4675	0.1693	0.4675	0.1685	0.4705
Map16	0.1689	0.4345	0.1689	0.4345	0.1922	0.6501
Map17	0.1646	0.4459	0.1646	0.4459	0.1646	0.4459
Map18	0.1764	0.5493	0.1753	0.4804	0.1751	0.4558
Map19	0.1782	0.4359	0.1782	0.4359	0.1782	0.4359
Map20	0.1732	0.5974	0.1732	0.5974	0.1726	0.4607
Map21	0.1697	0.4539	0.1697	0.4539	0.1697	0.4539
Map22	0.1648	0.4727	0.1648	0.4727	0.1648	0.4727
Map23	0.1717	0.4619	0.1717	0.4619	0.1718	0.4577
Map24	0.1850	0.4908	0.1850	0.4908	0.1830	0.4520
Map25	0.1695	0.4574	0.1695	0.4574	0.1695	0.4574
Map26	0.1739	0.4757	0.1739	0.4757	0.1739	0.4757

Table 4.7: Tracking Errors for Differential Drive Robot

Tracking Errors for Ackermann Steering Robot (meters)						
	K=1		**K=0.5**		**K=0.25**	
Map	Mean(m)	Max(m)	Mean(m)	Max(m)	Mean(m)	Max(m)
Map1	0.1920	0.4248	0.1920	0.4248	0.1920	0.4248
Map2	0.2795	0.6716	0.2795	0.6716	0.2795	0.6716
Map3	0.3318	0.7659	0.3318	0.7659	0.3271	0.7752
Map4	0.2193	0.6729	0.2193	0.6729	0.2193	0.6729
Map5	0.2559	0.7012	0.2559	0.7012	0.2550	0.6904
Map6	0.2616	0.7268	0.2616	0.7268	0.2641	0.7249
Map7	0.2494	0.7145	0.2494	0.7145	0.2494	0.7145
Map8	0.2778	0.6326	0.2778	0.6326	0.2778	0.6326
Map9	0.2305	0.5533	0.1974	0.5532	0.1974	0.5532
Map10	0.3745	0.7678	0.3745	0.7678	0.3805	0.7658
Map11	0.2689	0.6888	0.2689	0.6888	0.2689	0.6888
Map12	0.2448	0.8013	0.2448	0.8013	0.2390	0.7486
Map13	0.1818	0.4883	0.1818	0.4883	0.2748	0.8068
Map14	0.2366	0.6940	0.2366	0.6940	0.2374	0.6757
Map15	0.3539	3.4686	0.3539	3.4686	0.2450	0.7434
Map16	0.2629	0.7220	0.2629	0.7220	0.2435	0.7263
Map17	0.1959	0.4562	0.1959	0.4562	0.1959	0.4562
Map18	0.2371	0.7378	0.2418	0.7553	0.2481	0.7765
Map19	0.2749	0.7322	0.2749	0.7322	0.2749	0.7322
Map20	0.2824	0.7437	0.2824	0.7437	0.2796	0.7375
Map21	0.1835	0.4476	0.1835	0.4476	0.1835	0.4476
Map22	0.2193	0.4638	0.2193	0.4638	0.2193	0.4638
Map23	0.2394	0.7443	0.2394	0.7443	0.2396	0.7347
Map24	0.3276	0.7844	0.3276	0.7844	0.3271	0.7953
Map25	0.2219	0.5779	0.2219	0.5779	0.2219	0.5779
Map26	0.2535	0.6838	0.2535	0.6838	0.2535	0.6838

Table 4.8: Tracking Errors for Ackermann Steering Robot

In Table 4.9 and Table 4.10 minimum, mean and maximum velocities for a car-like robot with Ackermann steering and differential drive robot are shown. These robots are used with non-zero velocity. In these tables, the velocities of three different curvature values which are κ=1, κ=0.50, and κ=0.25, are shown as minimum, mean and maximum velocity parameters. As mentioned in Section 4.3, the effect of the curvature change varies according to the characteristic features of the map. This effect also showed itself in the velocities similar to tracking errors. Minimum, mean, and maximum velocities have also changed according to the changes in the path.

In table comparison, minimum and maximum velocities are similar for both differential drive and car-like robot with Ackermann steering. Since both robots are tested in the same environment, it is expected that the maximum velocities on flatter paths and the minimum velocities on more sharp corner's sections of the path are similar. For this reason, it would be more useful to look at the mean velocities in order to observe the curvature effect more clearly. Mean velocities are different for these robots. Generally, mean velocity of car-like robot with Ackermann steering is larger than the differential robot's mean velocity.

According to the characteristic features of the map, minimum, mean, and maximum velocity values are given in the tables. However, looking at the velocity changes as a plot instead of tables will be useful to understand the effect of the curvature value on the velocity. In this way, it can be seen more clearly with which velocity the robots follow the paths at the same curvature value.

Velocities for Differential Drive Robot (meter/second)									
	K=1			K=0.5			K=0.25		
Map	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**
Map 1	3.71	9.55	27.90	3.71	9.55	27.90	3.71	9.55	27.90
Map 2	2.40	5.78	30.86	2.40	5.78	30.86	2.40	5.78	30.86
Map 3	1.75	6.52	31.36	1.75	6.52	31.36	2.01	6.63	30.49
Map 4	2.33	9.35	30.84	2.33	9.35	30.84	2.33	9.35	30.84
Map 5	1.92	7.77	29.94	1.92	7.77	29.94	2.29	7.83	30.30
Map 6	1.89	6.02	31.04	1.89	6.02	31.04	2.32	6.17	31.04
Map 7	2.13	6.77	30.43	2.13	6.77	30.43	2.13	6.77	30.43
Map 8	2.51	5.04	27.22	2.51	5.04	27.22	2.51	5.04	27.22
Map 9	1.14	8.08	30.24	0.59	7.50	30.24	0.59	7.50	30.24
Map 10	1.82	4.10	27.46	1.82	4.10	27.46	2.00	4.14	30.21
Map 11	2.17	7.51	30.90	2.17	7.51	30.90	2.17	7.51	30.90
Map 12	1.52	6.89	30.85	1.52	6.89	30.85	2.08	6.40	30.85
Map 13	1.64	8.31	30.62	1.64	8.31	30.62	2.05	8.49	30.62
Map 14	1.90	8.29	30.85	1.90	8.29	30.85	2.40	8.30	30.85
Map 15	2.00	7.13	30.35	2.00	7.13	30.35	2.25	7.16	30.68
Map 16	1.69	6.27	31.22	1.69	6.27	31.22	1.96	6.47	31.22
Map 17	2.88	11.11	31.06	2.88	11.11	31.06	2.88	11.11	31.06
Map 18	1.20	9.49	31.53	1.57	9.44	29.44	2.03	9.45	30.79
Map 19	2.17	6.52	30.92	2.17	6.52	30.92	2.17	6.52	30.92
Map 20	1.76	6.32	29.57	1.76	6.32	29.57	2.16	6.32	28.63
Map 21	4.18	11.77	30.94	4.18	11.77	30.94	4.18	11.77	30.94
Map 22	3.03	8.05	31.05	3.03	8.05	31.05	3.03	8.05	31.05
Map 23	1.82	9.84	31.43	1.82	9.84	31.43	2.26	9.92	31.43
Map 24	1.73	4.82	30.67	1.73	4.82	30.67	2.02	4.93	30.67
Map 25	2.67	9.78	29.79	2.67	9.78	29.79	2.67	9.78	29.79
Map 26	2.25	9.21	31.37	2.25	9.21	31.37	2.25	9.21	31.37

Table 4.9: Velocities for Differential Drive Robot

Velocities for Car-Like Robot with Ackermann Steering (meter/second)									
	K=1			K=0.5			K=0.25		
Map	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**
Map 1	3.71	9.94	27.90	3.71	9.94	27.90	3.71	9.94	27.90
Map 2	2.40	6.93	30.86	2.40	6.93	30.86	2.40	6.93	30.86
Map 3	1.75	7.63	31.36	1.75	7.63	31.36	2.01	7.66	30.49
Map 4	2.33	9.96	30.84	2.33	9.96	30.84	2.33	9.96	30.84
Map 5	1.92	8.84	29.94	1.92	8.84	29.94	2.29	8.88	30.30
Map 6	1.89	6.79	31.04	1.89	6.79	31.04	2.32	6.95	31.04
Map 7	2.13	7.61	30.43	2.13	7.61	30.43	2.13	7.61	30.43
Map 8	2.51	5.42	27.22	2.51	5.42	27.22	2.51	5.42	27.22
Map 9	1.14	8.62	30.24	0.59	7.51	30.24	0.59	7.51	30.24
Map 10	1.82	5.03	27.46	1.82	5.03	27.46	2.00	4.98	30.21
Map 11	2.17	8.59	30.90	2.17	8.59	30.90	2.17	8.59	30.90
Map 12	1.52	7.88	30.85	1.52	7.88	30.85	2.08	7.33	30.85
Map 13	1.64	9.36	31.56	1.64	9.36	31.56	2.05	9.53	31.56
Map 14	1.90	8.83	30.85	1.90	8.83	30.85	2.40	8.81	30.85
Map 15	2.00	7.94	30.35	2.00	7.94	30.35	2.25	7.94	30.68
Map 16	1.69	7.00	31.22	1.69	7.00	31.22	2.05	7.22	31.22
Map 17	2.88	11.85	31.06	2.88	11.85	31.06	2.88	11.85	31.06
Map 18	2.16	10.98	31.53	2.08	10.79	29.44	2.03	10.65	30.79
Map 19	2.17	7.21	30.92	2.17	7.21	30.92	2.17	7.21	30.92
Map 20	1.76	7.22	29.57	1.76	7.22	29.57	2.16	7.18	28.63
Map 21	4.18	12.30	30.94	4.18	12.30	30.94	4.18	12.30	30.94
Map 22	3.03	8.68	31.05	3.03	8.68	31.05	3.03	8.68	31.05
Map 23	1.82	10.79	31.43	1.82	10.79	31.43	2.26	10.74	31.43
Map 24	1.73	5.56	30.67	1.73	5.56	30.67	2.02	5.71	30.67
Map 25	2.67	10.48	29.79	2.67	10.48	29.79	2.67	10.48	29.79
Map 26	2.25	10.37	31.37	2.25	10.37	31.37	2.25	10.37	31.37

Table 4.10: Velocities for Car-Like Robot with Ackermann Steering

There are three different velocity vs time plot are given for different maps to understand difference between differential drive and car-like robot with Ackermann steering. In Fig. 4.27, sparse map example is given for Map 1. Plots are similar for both robots as a maximum and minimum velocities. Main difference occurs at maximum velocity parts. Blue lines which are used for differential drive have more sharp changes at maximum velocities. The change at the velocity occurs more quick than the car-like robot with Ackermann steering. For instance, car like robot stays at maximum velocity a bit longer than the differential drive robot.

In Fig. 4.28 and Fig. 4.29 two different plots are given with more different mean velocities for both type of robots. Similar to the Fig. 4.27, general plots are similar. Main changes occur at deceleration and acceleration parts. Generally, car-like robots accelerate and decelerate earlier than the differential drive robot.

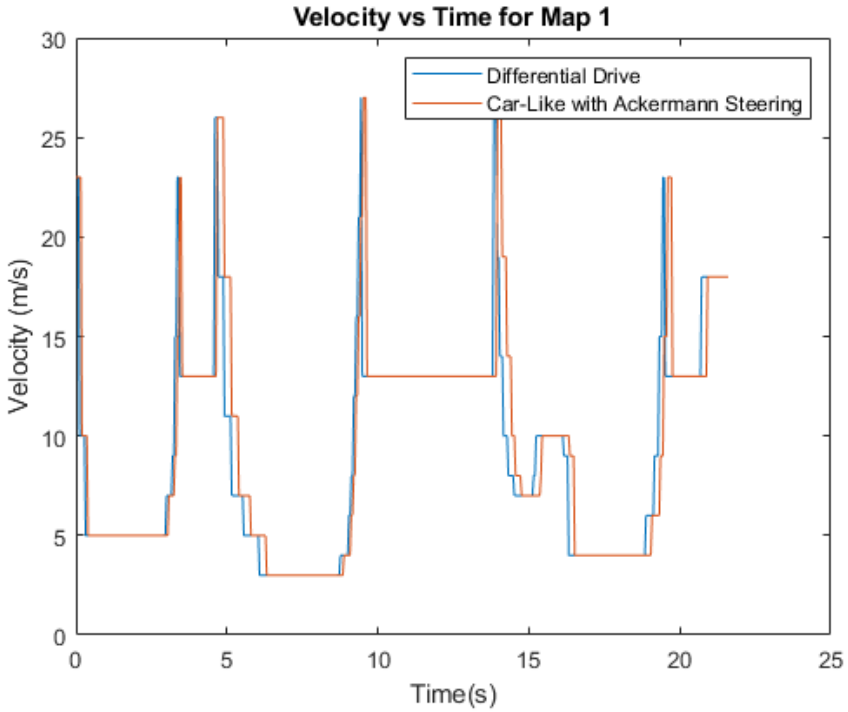

Figure 4.27: Velocity Differences for Map 1 with $\kappa = 0.25$

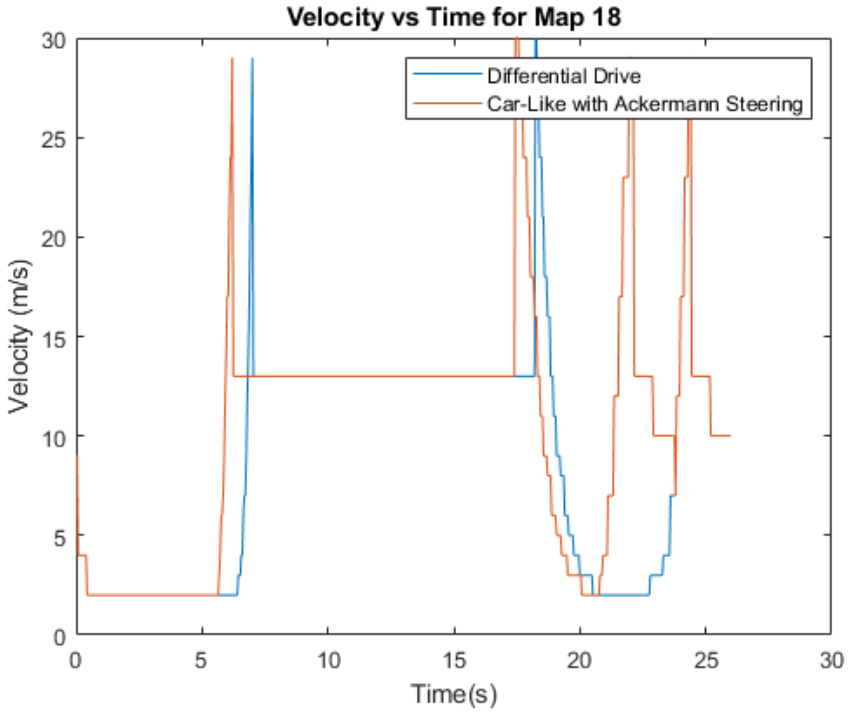

Figure 4.28: Velocity Differences for Map 18 with $\kappa = 0.25$

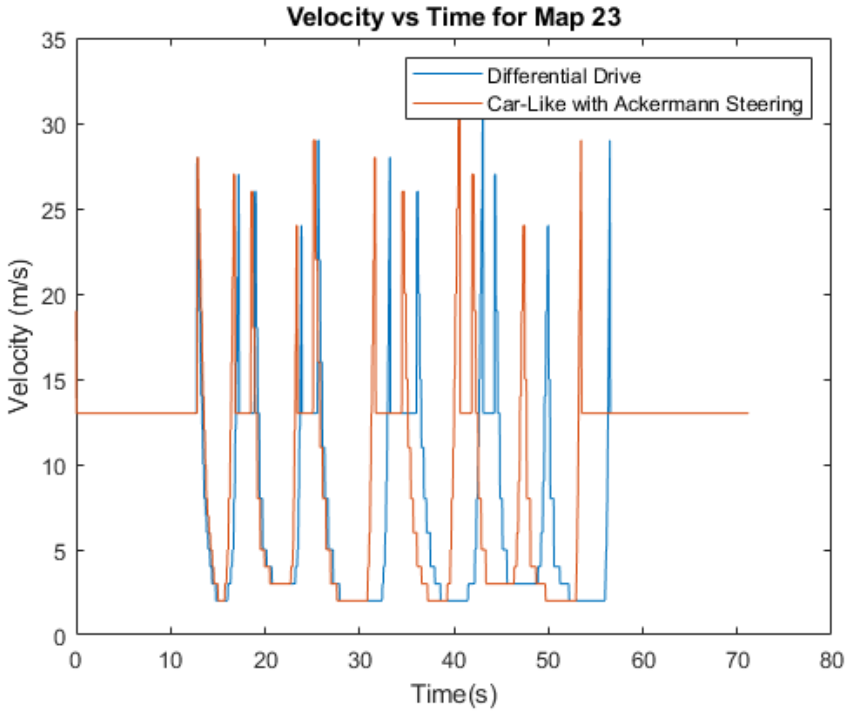

Figure 4.29: Velocity Differences for Map 23 with $\kappa = 0.25$

CHAPTER 5

CONCLUSION

This thesis explains research on path planning and tracking for nonholonomic robots such as differential drive robots or car-like robots with Ackermann steering. These robots are a field of research that is developing day by day and path planning and tracking for such robots has great importance. It is essential that the planned path is safe, as short as possible, and can be followed by the vehicle. It is preferred to include smooth curves instead of sharp corners so that the planned path can be followed by a non-holonomic robot.

In this thesis, the generation of smooth curves instead of sharp corners is made with Bezier curves. The proposed algorithm includes the following steps; merging close waypoints obtained from VV-ST-R algorithm's waypoints, moving waypoints away from the obstacle, generating helper control points for Bezier curve, inserting new control points for Bezier curve if Bezier curve segments hit obstacle, and finally creation of Bezier curves with six control points. Other than that, to handle the curvature constraints, if the Bezier curve segment's curvature is larger than maximum curvature value, the locations of the helper control points are changed to obtain desired curvature.

The developed method is applied based on VB and VV-ST-R path planning algorithms that generate straight-line paths with sharp corners. For the comparison parameters path length, minimum distance to the obstacle, and computation time are selected. For the path length and minimum distance to the obstacle parameters results are similar. Since the VB path aims that the safest path its minimum distance to the obstacle and path length is the largest. Since the VV-ST-R path aims that the shortest path its minimum distance to the obstacle and path length is the smallest. The pro-

posed algorithm uses control points generated from VV-ST-R waypoints. However, it changes waypoint locations away from the obstacle to find smooth curves. By doing these changes, the proposed algorithm uses a safe region between the VB path and the VV-ST-R path. Therefore, the path length and minimum distance to the obstacle parameters are between two other algorithms. For the computation time parameter, since previous algorithms are assumed to be calculated, only the computation time between different maps is compared. The proposed algorithm's computation time varies according to the number of control points on the path. If there is a path on the map that needs to be passed around detailed obstacles, the computation time is longer. This is because of many control points are necessary to give direction around the obstacles. If the distance between the obstacles on the map is large and there is no need for more control points to find the way, the computation time is shorter.

Curvature constraint is another parameter for comparison. A small enough curvature is necessary to ensure that nonholonomic robots can follow these generated smooth paths with their kinematic constraints. To handle the curvature constraints another method is applied to the proposed algorithm, changing helper control points. While changing control points locations path length is also changed. To handle curvature constraints, the proposed algorithm places helper control points from each other. Thus, the resulting Bezier curve becomes more smooth. This causes an extension in the path length. Therefore, the path length becomes larger when small curvature is wanted. The minimum distance to obstacle value decreased as the curvature value decreased. This has been observed because the algorithm brings the path closer to the far corners of the corridors to reach the desired curvature value.

To further study the path quality, the Pure Pursuit algorithm was used for the traceability of smooth paths by car-like robots with Ackermann steering and differential drive robots. It was observed that the path found by the proposed algorithm could follow smooth paths by the robots. However, the overshoot was observed in non-smooth paths. Therefore, it was observed that the smooth paths generated in cases with control points away from the obstacle had higher traceability by vehicles.

REFERENCES

[1] I. Nourbakhsh, R. Powers, and S. Birchfield, "Dervish an office-navigating robot," *AI Magazine*, vol. 16, p. 53, Jun. 1995.

[2] C. Zhang, D. Chu, S. Liu, Z. Deng, C. Wu, and X. Su, "Trajectory planning and tracking for autonomous vehicle based on state lattice and model predictive control," *IEEE Intelligent Transportation Systems Magazine*, vol. 11, no. 2, pp. 29–40, 2019.

[3] M. R. H. Al-Dahhan and K. W. Schmidt, "Voronoi boundary visibility for efficient path planning," *IEEE Access*, vol. 8, pp. 134764–134781, 2020.

[4] J. R. Sánchez-Ibáñez, C. J. Pérez-del Pulgar, and A. García-Cerezo, "Path planning for autonomous mobile robots: A review," *Sensors*, vol. 21, no. 23, 2021.

[5] O. Sharma, N. C. Sahoo, and N. B. Puhan, "A survey on smooth path generation techniques for nonholonomic autonomous vehicle systems," in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, pp. 5167–5172, 2019.

[6] K. Kawabata, L. Ma, J. Xue, C. Zhu, and N. Zheng, "A path generation for automated vehicle based on bezier curve and via-points," *Robotics and Autonomous Systems*, vol. 74, pp. 243–252, 2015.

[7] A. Ravankar, A. A. Ravankar, Y. Kobayashi, and T. Emaru, "Path smoothing extension for various robot path planners," in *2016 16th International Conference on Control, Automation and Systems (ICCAS)*, pp. 263–268, 2016.

[8] K. Yang, D. Jung, and S. Sukkarieh, "Continuous curvature path-smoothing algorithm using cubic b zier spiral curves for non-holonomic robots," *Advanced Robotics*, vol. 27, no. 4, pp. 247–258, 2013.

[9] A. Ravankar, A. A. Ravankar, Y. Kobayashi, Y. Hoshino, and C.-C. Peng, "Path

smoothing techniques in robot navigation: State-of-the-art, current and future challenges," *Sensors*, vol. 18, no. 9, 2018.

[10] A. Woo, B. Fidan, W. Melek, S. Zekavat, and R. Buehrer, "Localization for autonomous driving," pp. 1051–1087, 01 2019.

[11] G. Oriolo, G. Ulivi, and M. Vendittelli, "Real-time map building and navigation for autonomous robots in unknown environments," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 28, no. 3, pp. 316–333, 1998.

[12] N. Sariff and N. Buniyamin, "An overview of autonomous mobile robot path planning algorithms," in *2006 4th Student Conference on Research and Development*, pp. 183–188, 2006.

[13] R. Gutiérrez, E. López-Guillén, L. M. Bergasa, R. Barea, Ó. Pérez, C. Gómez-Huélamo, F. Arango, J. Del Egido, and J. López-Fernández, "A waypoint tracking controller for autonomous road vehicles using ros framework," *Sensors*, vol. 20, no. 14, p. 4062, 2020.

[14] J. E. da Silva and J. B. de Sousa, "A dynamic programming approach for the motion control of autonomous vehicles," in *49th IEEE Conference on Decision and Control (CDC)*, pp. 6660–6665, 2010.

[15] R. M. C. Santiago, A. L. De Ocampo, A. T. Ubando, A. A. Bandala, and E. P. Dadios, "Path planning for mobile robots using genetic algorithm and probabilistic roadmap," in *2017IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, pp. 1–5, 2017.

[16] S. Li, Z. Li, Z. Yu, B. Zhang, and N. Zhang, "Dynamic trajectory planning and tracking for autonomous vehicle with obstacle avoidance based on model predictive control," *IEEE Access*, vol. 7, pp. 132074–132086, 2019.

[17] S. Dixit, S. Fallah, U. Montanaro, M. Dianati, A. Stevens, F. Mccullough, and A. Mouzakitis, "Trajectory planning and tracking for autonomous overtaking: State-of-the-art and future prospects," *Annual Reviews in Control*, vol. 45, pp. 76–86, 2018.

[18] P. Wang, S. Gao, L. Li, B. Sun, and S. Cheng, "Obstacle avoidance path planning design for autonomous driving vehicles based on an improved artificial potential field algorithm," *Energies*, vol. 12, no. 12, 2019.

[19] H. Li, C. Wu, D. Chu, L. Lu, and K. Cheng, "Combined trajectory planning and tracking for autonomous vehicle considering driving styles," *IEEE Access*, vol. 9, pp. 9453–9463, 2021.

[20] T. Maekawa, T. Noda, S. Tamura, T. Ozaki, and K. ichiro Machida, "Curvature continuous path generation for autonomous vehicle using b-spline curves," *Computer-Aided Design*, vol. 42, no. 4, pp. 350–359, 2010.

[21] R. Wang, Y. Li, J. Fan, T. Wang, and X. Chen, "A novel pure pursuit algorithm for autonomous vehicles based on salp swarm algorithm and velocity controller," *IEEE Access*, vol. 8, pp. 166525–166540, 2020.

[22] L. Han, H. Yashiro, H. Tehrani Nik Nejad, Q. H. Do, and S. Mita, "Bézier curve based path planning for autonomous vehicle in urban environment," in *2010 IEEE Intelligent Vehicles Symposium*, pp. 1036–1042, 2010.

[23] N. B. A. Latip and R. Omar, "Feasible path generation using bezier curves for car-like vehicle," *IOP Conference Series: Materials Science and Engineering*, vol. 226, p. 012133, aug 2017.

[24] J.-w. Choi, R. E. Curry, and G. H. Elkaim, "Curvature-continuous trajectory generation with corridor constraint for autonomous ground vehicles," in *49th IEEE Conference on Decision and Control (CDC)*, pp. 7166–7171, 2010.

[25] I. Makarov and P. Polyakov, "Smoothing voronoi-based path with minimized length and visibility using composite bezier curves," in *AIST*, 2016.

[26] Y.-J. Ho and J.-S. Liu, "Collision-free curvature-bounded smooth path planning using composite bezier curve based on voronoi diagram," in *2009 IEEE International Symposium on Computational Intelligence in Robotics and Automation - (CIRA)*, pp. 463–468, 2009.

[27] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

[28] T. Marques and V. Silva, “Generating triangular lattices for surfaces with irregular boundary,” pp. 511–525, 01 2016.

[29] “Path following for a differential drive robot.” https://www.mathworks.com/help/robotics/ug/path-following-for-differential-drive-robot.html/. Accessed: 2022-09-08.

[30] C. Gamez Serna, A. Lombard, Y. Ruichek, and A. Abbas-Turki, “Gps-based curve estimation for an adaptive pure pursuit algorithm,” pp. 497–511, 08 2017.

Appendix A

PROPOSED ALGORITHMS RESULTS

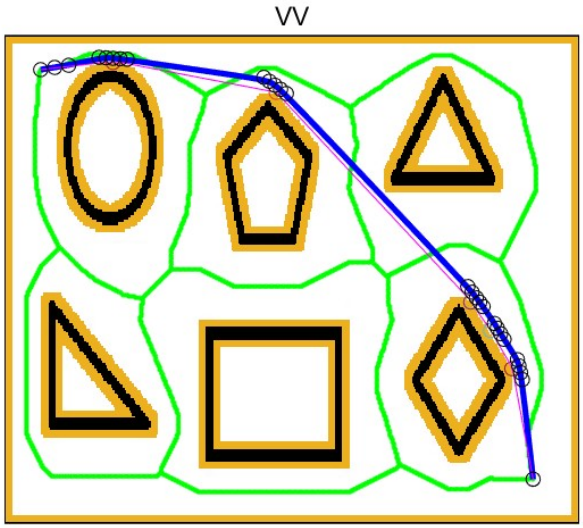

Figure A.1: Obtained Smooth Path for Map 1

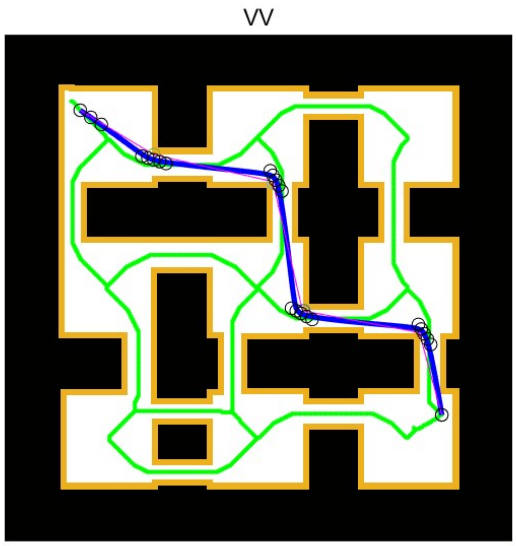

Figure A.2: Obtained Smooth Path for Map 2

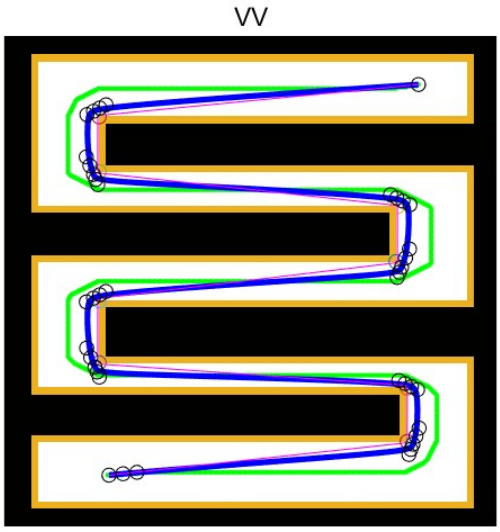

Figure A.3: Obtained Smooth Path for Map 3

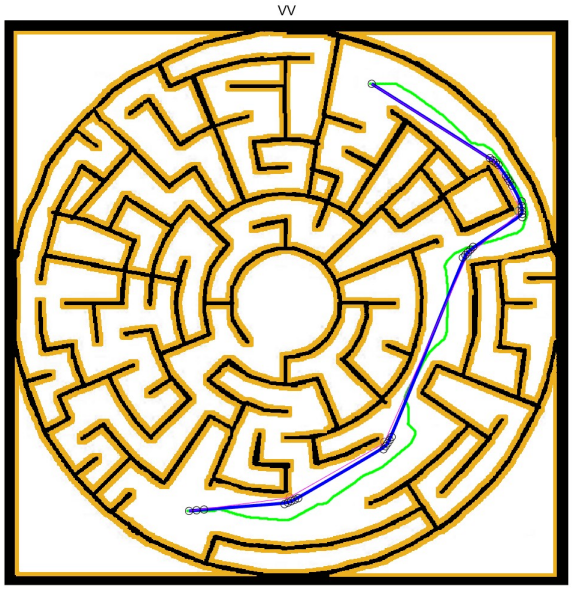

Figure A.4: Obtained Smooth Path for Map 4

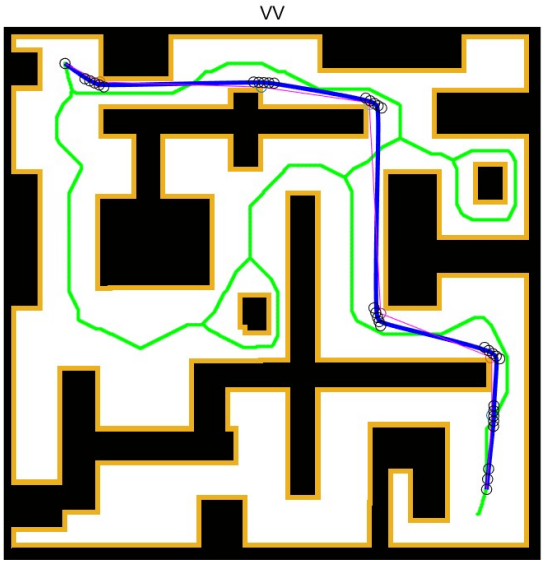

Figure A.5: Obtained Smooth Path for Map 5

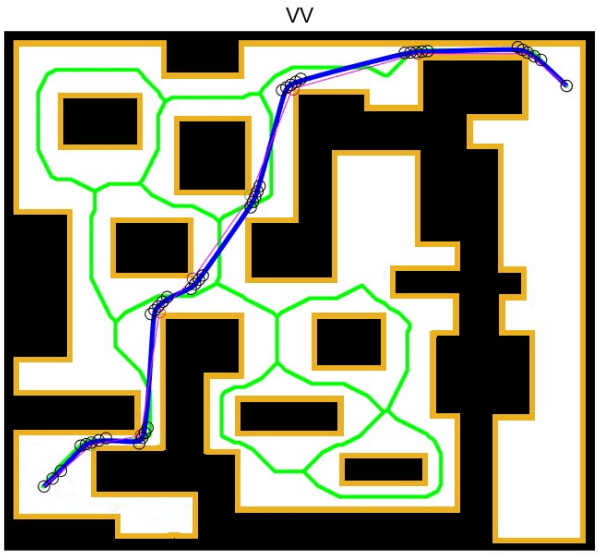

Figure A.6: Obtained Smooth Path for Map 6

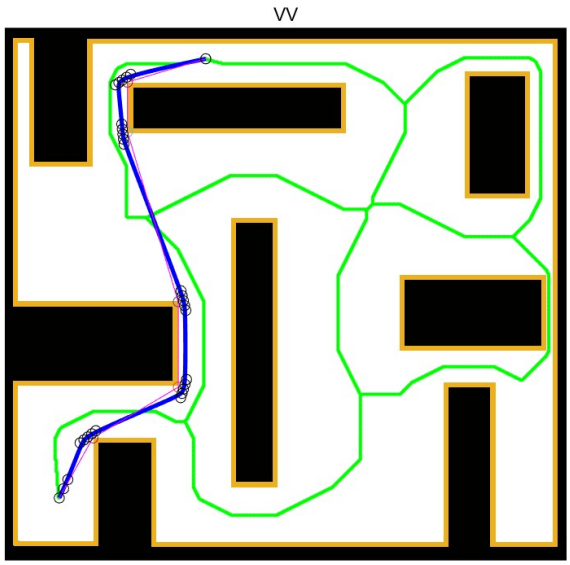

Figure A.7: Obtained Smooth Path for Map 7

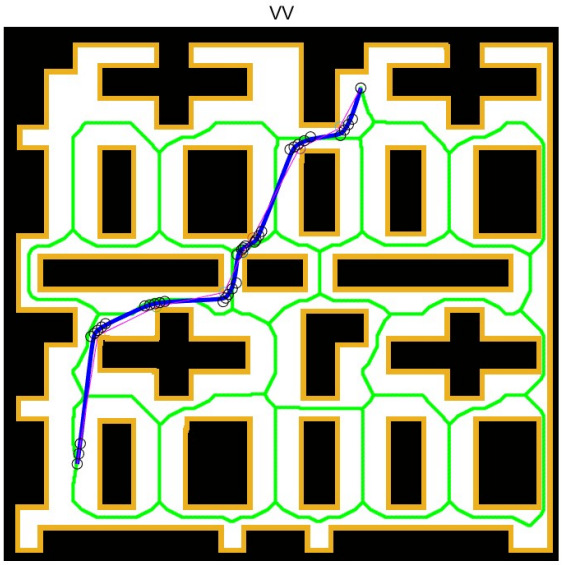

Figure A.8: Obtained Smooth Path for Map 8

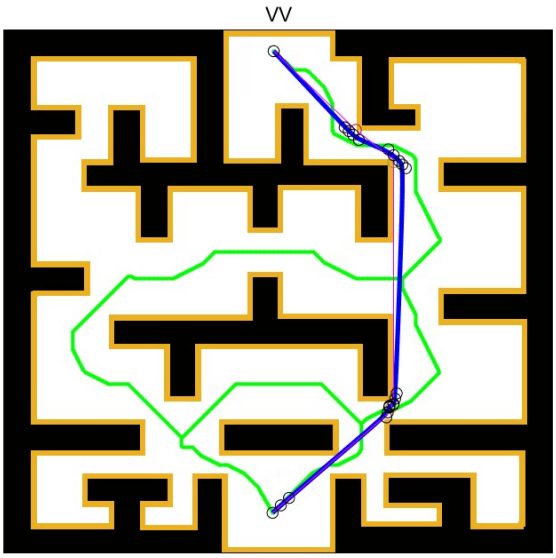

Figure A.9: Obtained Smooth Path for Map 9

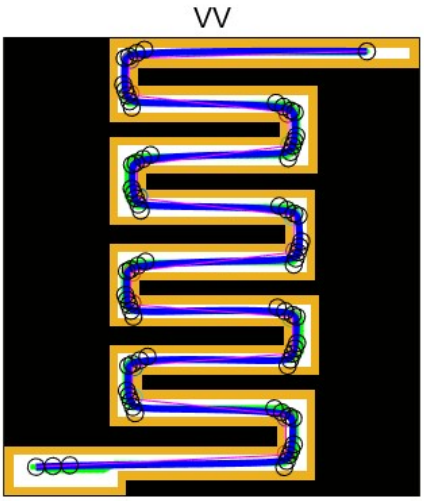

Figure A.10: Obtained Smooth Path for Map 10

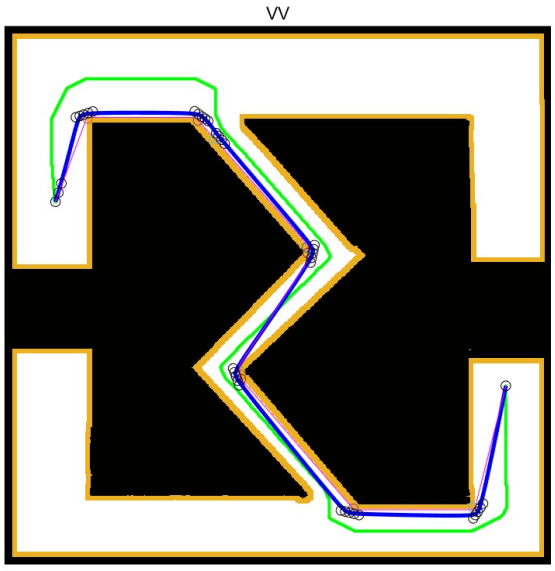

Figure A.11: Obtained Smooth Path for Map 11

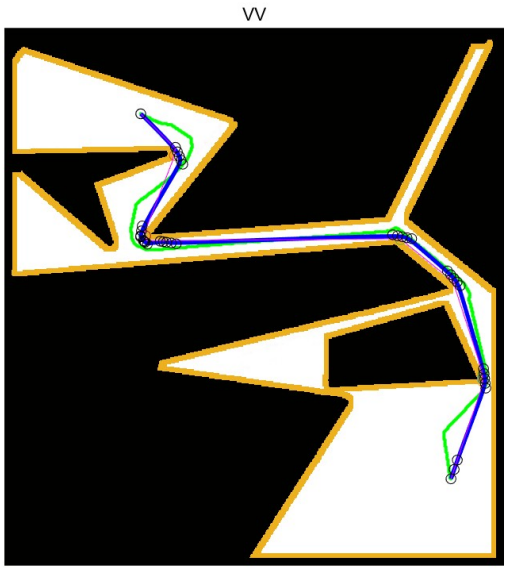

Figure A.12: Obtained Smooth Path for Map 12

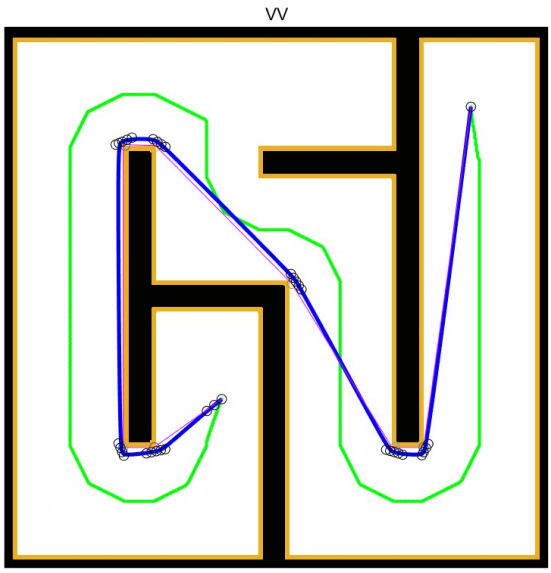

Figure A.13: Obtained Smooth Path for Map 13

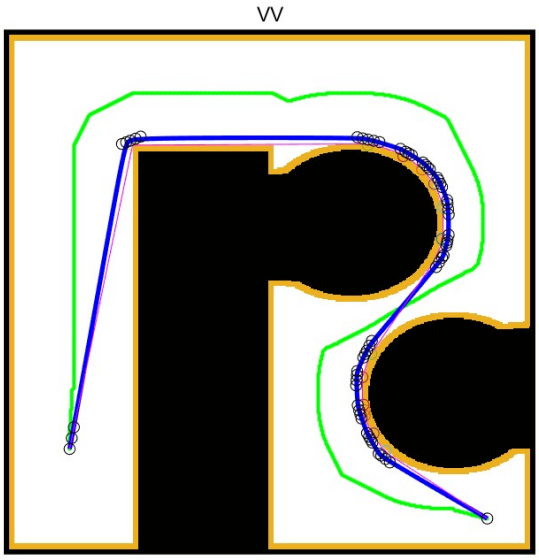

Figure A.14: Obtained Smooth Path for Map 14

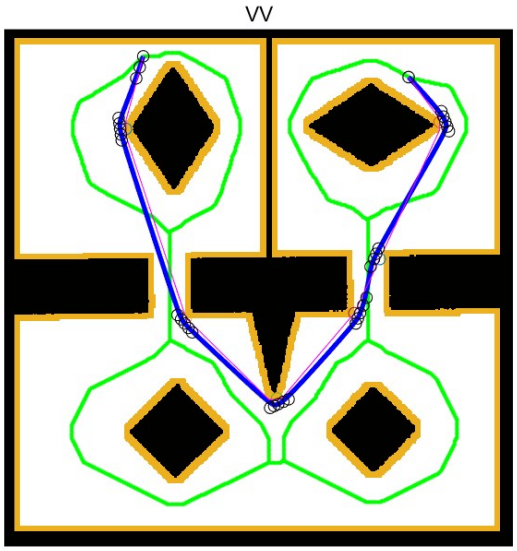

Figure A.15: Obtained Smooth Path for Map 15

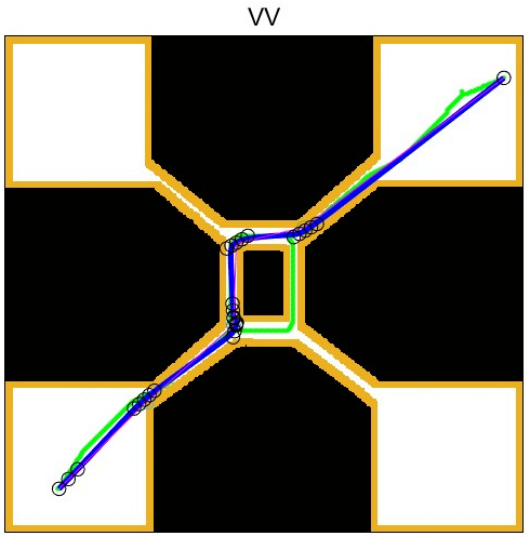

Figure A.16: Obtained Smooth Path for Map 16

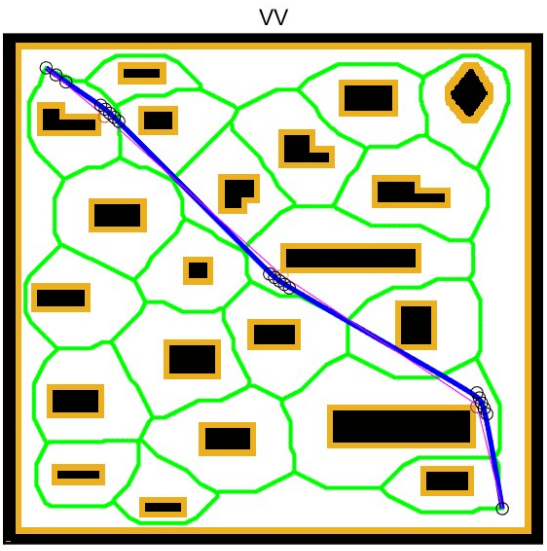

Figure A.17: Obtained Smooth Path for Map 17

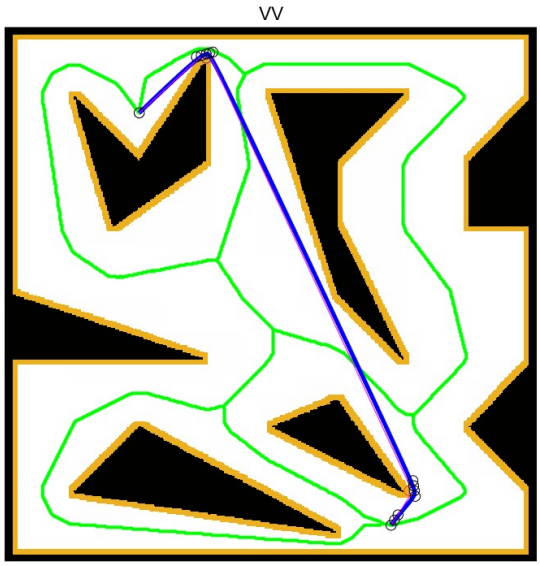

Figure A.18: Obtained Smooth Path for Map 18

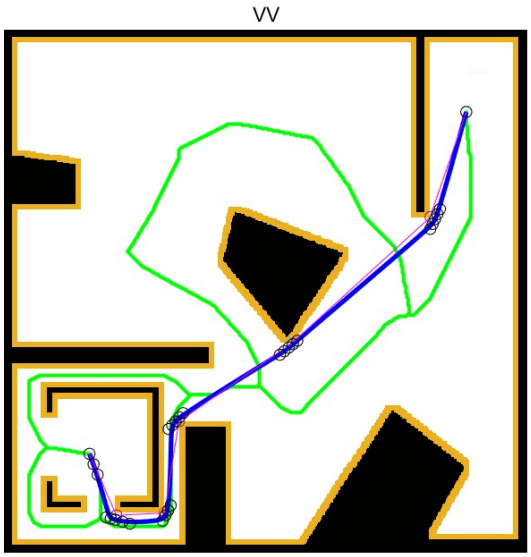

Figure A.19: Obtained Smooth Path for Map 19

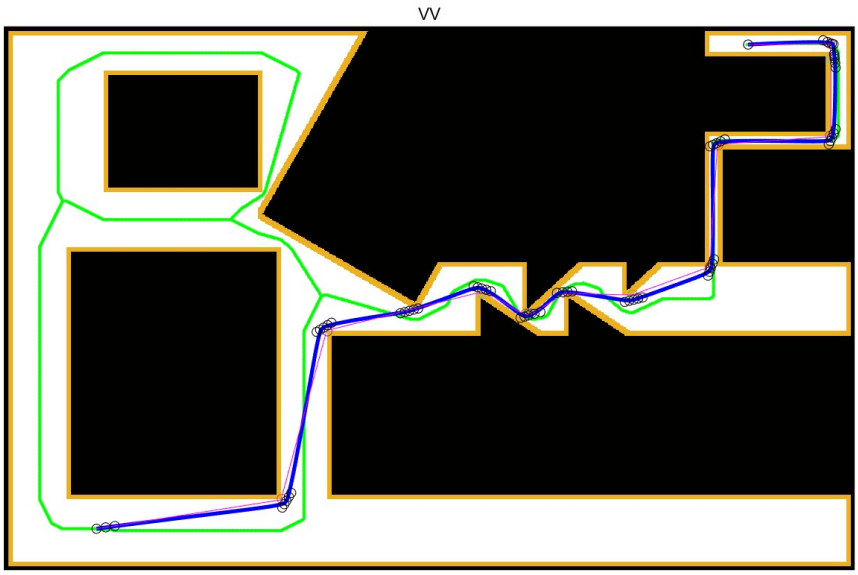

Figure A.20: Obtained Smooth Path for Map 20

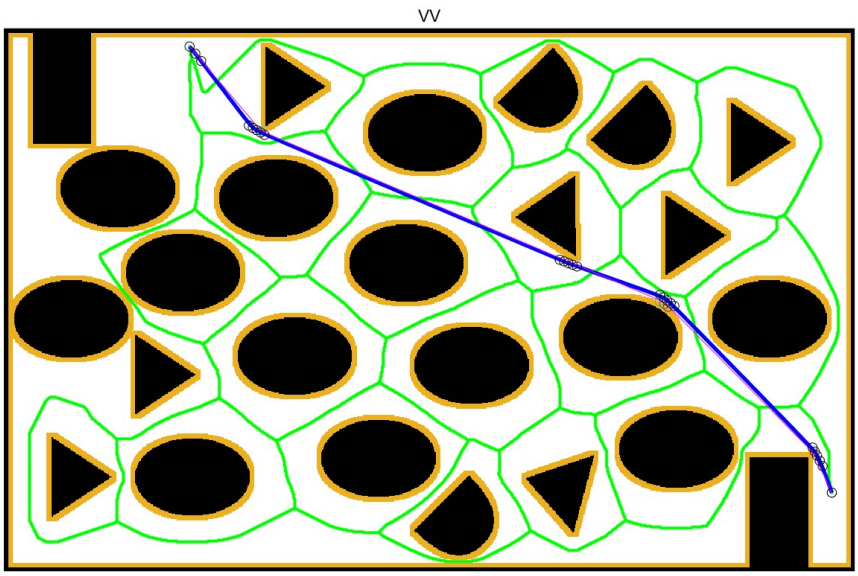

Figure A.21: Obtained Smooth Path for Map 21

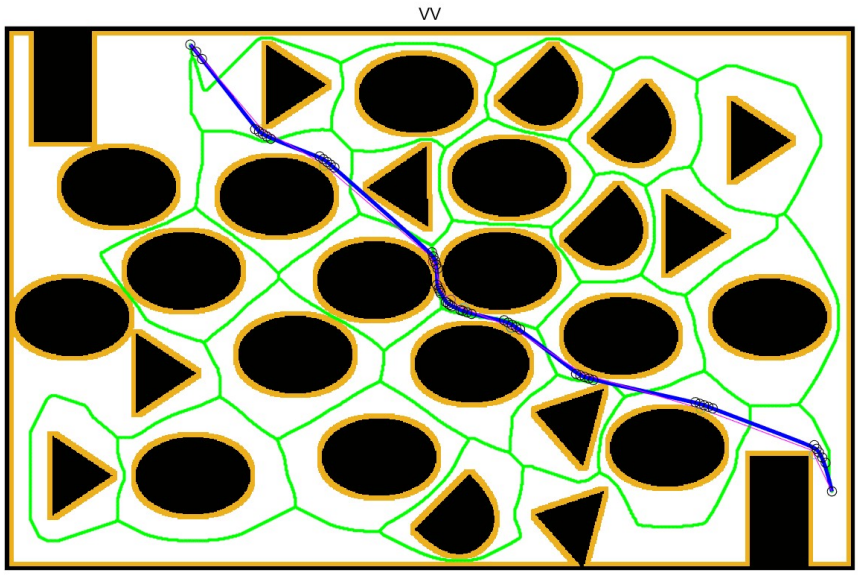

Figure A.22: Obtained Smooth Path for Map 22

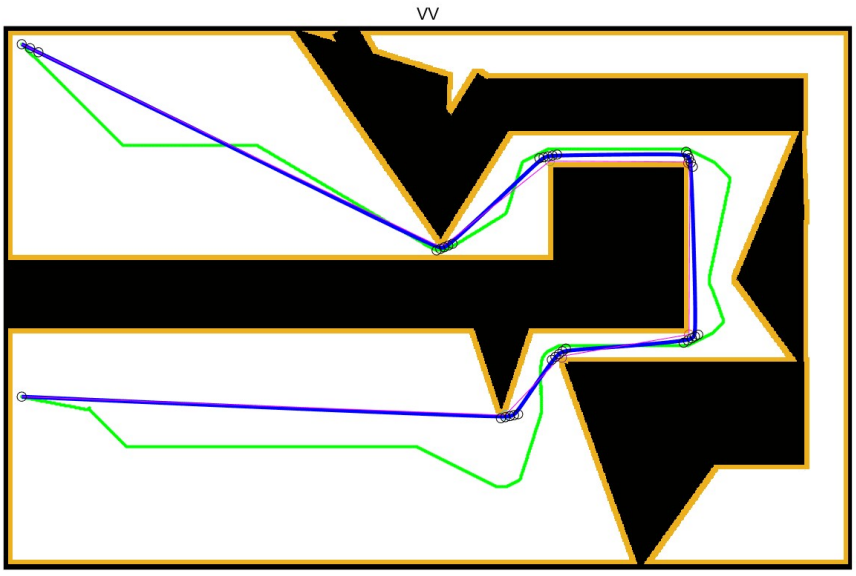

Figure A.23: Obtained Smooth Path for Map 23

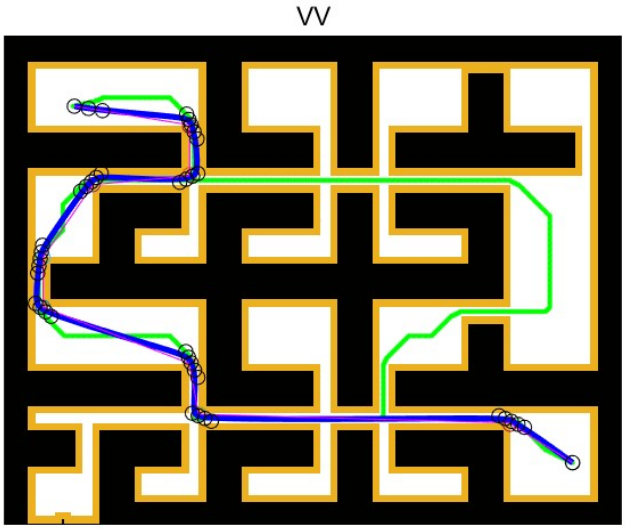

Figure A.24: Obtained Smooth Path for Map 24

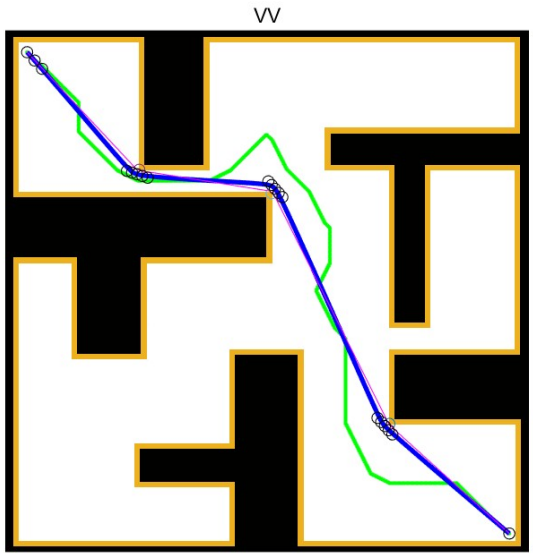

Figure A.25: Obtained Smooth Path for Map 25

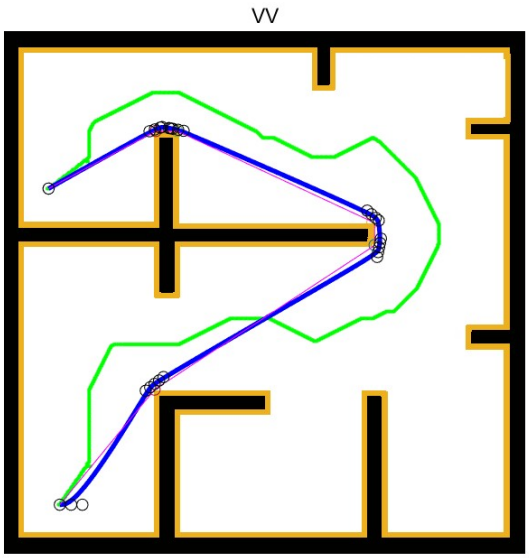

Figure A.26: Obtained Smooth Path for Map 26