Trajectory generation for road vehicles using Bezier curves and non-linear optimisation

Jai Sabnis

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2023

Abstract

Collision-free trajectory generation is a critical aspect of autonomous driving systems that ensures the safe operation of vehicles. The primary goal of trajectory generation is to generate paths for the vehicle that avoid collisions with obstacles and other vehicles in the environment. This study will explore the use of bezier curves along with SQP optimisation in producing smooth and collision-free trajectories for road vehicles.

While prior explorations have focused on using bezier curves for trajectory generation for simplistic road scenarios involving single obstacles and well-defined roads, this paper builds upon prior methods and adapts them to work for more challenging scenarios involving multiple obstacles and roads without lanes. The proposed method is then tested for a variety of test scenarios and its performance is evaluated. Thus, this paper presents a promising approach for trajectory generation for autonomous vehicles using bezier curves.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Jai Sabnis*)

Acknowledgements

I would like to sincerely thank my supervisor, Steve Tonneau, for his patience, time and constant support. There have been many instances where he has gone out of his way to help me with my project and I am truly grateful for that.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

The development of collision-free trajectory generation for road vehicles is an essential area of research in the field of autonomous driving. This technology has the potential to revolutionize the transportation industry by making road travel safer, more efficient, and more accessible to everyone [22].

Currently, road accidents are one of the leading causes of death and injury worldwide [16]. The majority of these accidents occur due to human error [13], such as distracted driving or misjudging distances. Autonomous driving technology, with the ability to generate collision-free trajectories, could drastically reduce the number of accidents caused by human error.

Moreover, the implementation of autonomous driving technology can also provide a solution to the issue of traffic congestion. Traffic congestion is a growing concern in urban areas, leading to increased travel times, fuel consumption, and air pollution. Autonomous vehicles, with the capability to communicate with each other and adjust their speeds and routes in real time, can help alleviate traffic congestion by optimizing traffic flow and minimizing traffic jams [1].

Another motivation for this research is to improve the accessibility of transportation for individuals with disabilities or who are unable to drive. Autonomous vehicles with collision-free trajectory generation technology can provide individuals with mobility impairments with the freedom and independence to travel without the need for a driver [2].

Most prior research in this field focuses on road travel in developed countries [26] with models that emulate the road conditions in these countries. Assumptions are made regarding the presence of lanes and the behaviour of obstacles. However, road conditions in developing countries such as India and Nigeria are vastly different. It is common for roads to not have structured lanes and for the presence of many stationary or slow-moving obstacles such as pedestrians, street vendors, animals and parked vehicles as shown in figure 1.1. My goal for this project was to develop a trajectory generation

(a) India [20]

(b) Nigeria [12]

Figure 1.1: Road conditions in India and Nigeria

method that would be better suited for such road conditions and then test the trajectory generation for scenarios that resembled these road conditions.

1.2 Objectives

The main objective of the project was to use bezier curves to develop and test a trajectory generation method that could effectively produce collision-free trajectories for a road vehicle for a variety of challenging scenarios. Another important objective was guaranteeing that generated trajectories would ensure safety and comfort for passengers inside the vehicle.

The resulting mathematical problem would be finding the optimal bezier curve trajectory that avoids collisions with all the obstacles on the road using non-convex optimisation. In this case, an optimal trajectory is one that is collision-free and also smooth i.e. characterised by low curvature.

The challenge associated with this problem comes from the unpredictable nature of the obstacles on the road since obstacles can be present at random positions and can move unpredictably. Avoiding collisions with these obstacles while also conforming to the limitations of the road and vehicle model adds considerable complexity to the problem.

1.3 Contributions

- Developed trajectory generation method for road vehicles that finds a smooth collision-free trajectory using non-linear optimisation.
  - Developed potential field function for roads that accounts for roads with and without lanes.
  - Modified existing optimisation problem for finding the optimal trajectory to include weights for curvature and potential field.
  - Developed alternative heuristic method for trajectory generation and tested its performance against optimized algorithm.
- Incorporated sliding window control to allow for trajectory generation that reacts to changes in the environment such as moving obstacles.
- Used optimized algorithm for generating trajectories for multiple vehicles on the same road.
- Tested and evaluated the performance of the trajectory generation on challenging test scenarios involving multiple stationary and moving obstacles.

Chapter 2

Background

2.1 Brief explanation of problem

The input of the problem will be information regarding the road conditions such as it's width, length and centre lines of lanes (if they are present) as well as the positions of obstacles on the road. The input will also include the vehicle information for the ego vehicle i.e. the vehicle we would like to generate a trajectory for. The information will include data such as the ego vehicle's starting position, the desired end point, its maximum steering angle, wheelbase and the desired velocity.

The method will then generate a collision-free trajectory from the start point to the desired end point in the form of a bezier curve for the ego vehicle on the road based on the road conditions while ensuring that the trajectory conforms to the constraints of the vehicle model.

2.2 Restrictions and Complexity associated with the problem

2.2.1 Restrictions associated with road vehicles

- Non-Holonomic Constraints and restrictions: Road vehicles have non-holonomic constraints [23], which means that they cannot instantaneously change their direction of motion. A road vehicle typically turns by pivoting around its centre of gravity and following a curved path. For instance, while a quadcopter can move at a 90-degree angle from its current direction of motion instantly- that is infeasible for a car or motorbike. This restriction limits the vehicle's manoeuvrability and makes it more difficult to generate trajectories that avoid obstacles.
- Complex Dynamics: Road vehicles have complex dynamics that depend on various factors, such as the vehicle's weight, size, shape, and tire properties. These dynamics make it challenging to predict the vehicle's motion accurately and generate collision-free trajectories.

2.2.2 Collision avoidance

The positions of an obstacle may change over time, which makes it challenging to ensure collision avoidance since the trajectory generation has to take this change into account. Another factor that can make collision avoidance difficult is the presence of multiple obstacles. In such cases, it may not be possible to simply navigate around each obstacle individually, and it may be necessary to find a path that avoids all of them simultaneously, which can be a complex optimization problem.

Prior research [28] in using bezier curves for trajectory generation has predominantly focused on avoiding just one obstacle at a time and assumes the obstacle moves predictably.

2.2.3 Efficient real-time path generation

The trajectories also need to be generated in real-time to ensure the safety of the vehicle and other road users. The time constraint here depends on the speed of the vehicle - higher speeds result in shorter time constraints but it is usually a fraction of a second. Other systems using algorithms such as RRT (Rapidly-exploring Random Tree) have been capable of generating trajectories for vehicles with maximum speeds of 0.5 m/s. [11]. This real-time constraint limits the complexity of the collision avoidance algorithms that can be used and requires a balance between accuracy and efficiency. Overall, the trajectory generation process must be fast enough to ensure that the vehicle can react to changing environmental conditions and avoid collisions.

2.2.4 Generation of a smooth path

Paths for road vehicles generated need to be smooth i.e. its derivatives have to be continuous. This means that the path has no sudden jumps or changes in direction, and it flows seamlessly from one point to another. This is required for several reasons:

- Comfort: Smooth trajectories help to ensure a comfortable ride for passengers by reducing sudden changes in velocity or direction. This is especially important for public transportation systems such as buses, where passengers are often standing and need to maintain their balance.
- Safety: Abrupt changes in trajectory can lead to instability and loss of control of the vehicle, which can increase the risk of accidents. Smooth trajectories help to ensure that the vehicle maintains a stable and predictable motion.

  This adds another layer of complexity since generated paths cannot include extremely sharp turns while avoiding collisions since they need to be smooth to ensure safety and comfort.

2.3 Methods for trajectory generation

While there are several methods for trajectory generation, this paper will predominantly focus on a hybrid model for trajectory generation that combines aspects of rule-based

generation with a model-based generation technique, specifically a probabilistic model.

2.3.1 Rule-based generation

Rule-based trajectory generation involves defining a set of rules that the vehicle must follow in order to generate its trajectory [10]. These rules could include points like obeying traffic signals, staying within speed limits, and following the correct lane markings. These rules are usually based on a set of predefined conditions and actions, such as ”if the traffic light is red, stop” or ”if a vehicle is in the way, slow down or change lanes.” While rule-based trajectory generation is generally simpler and easier to implement, it may not always result in the most efficient or optimal trajectory.

2.3.2 Model-based generation

Model-based trajectory generation involves creating a mathematical model of the vehicle’s motion and using it to generate trajectories that are both safe and efficient [9]. This model takes into account various factors such as the vehicle’s acceleration, speed, and turning radius, as well as environmental factors such as the dimensions of the road and other vehicles. The model can then generate trajectories that are optimized for variables like fuel efficiency, travel time, and path smoothness. Model-based trajectory generation can produce more optimal trajectories, but it requires a more complex mathematical model and may be more computationally intensive.

2.3.3 Probabilistic model

This paper will employ a probabilistic model for generating trajectories. A probabilistic model is used to estimate the likelihood of different outcomes, such as the likelihood of reaching the target position and the likelihood of encountering obstacles along the way. These probabilities are then incorporated into the trajectory generation process, where they are used to generate safe and efficient trajectories that take into account the uncertainty in the environment [27].

To generate the trajectory, the technique first defines a set of candidate paths between the current position and the target position. Each path is assigned a probability based on the likelihood of a collision, which takes into account factors such as the position of the ego vehicle and other vehicles, the dimensions of the road, etc. After this, it is a matter of finding the most optimal path.

2.4 Prior work

Considerable work has already been done in this topic area of using Bezier curves for path generation for autonomous vehicles.

2.4.1 Continuous Curvature Path Generation Based on Bézier Curves for Autonomous Vehicles. (2010)

In this paper [6], Bezier curves were used to generate a continuous path by placing bezier curves within segments to allow for smoother turns. Researchers also compared these paths to those generated using circular arcs and line segments. While the circular arcs seemed a good option because of their efficiency in terms of ease of computation they resulted in paths requiring moving in an infeasible manner or paths that went out of bounds. Whereas bezier curves while slightly less efficient provided safer trajectories.

2.4.2 Lane Change Path Planning Based on Piecewise Bezier Curve for Autonomous Vehicle (2013)

While paper [6] was able to generate smooth trajectories- these trajectories did not account for any obstacles in the path. In order to include capabilities such as obstacle avoidance, this research paper [5] used piecewise Bezier curves to add rule-based functionality for lane changing wherein the autonomous vehicle can switch lanes to find a more optimal path if required.

In this solution, the vehicle will ensure that there is a safe distance between itself and the vehicle in front of it and accordingly change lanes. The method would generate a path using a third-order bezier curve that can be followed smoothly ensuring the ride-comfort of the passengers in the car. However, this method only takes into account the possibility of collisions with the vehicle in front of it and doesn't factor in possible collisions in other lanes.

2.4.3 Path generation and tracking based on a Bezier curve for a steering rate controller of autonomous vehicles (2013)

Building upon the ideas proposed by the prior papers, researchers for this paper [3] proposed using fifth-order Bezier curves (curves defined by a set of five control points) instead of the third-order Bezier curves used by Choi [6] to improve the smoothness of the path generated while lane changing and turning. They found that when the lane change was implemented using third-order Bezier curves - as described by Chen [5]- the angle of the heading was not the same at the start and endpoints. This results in obvious problems on a straight road since it means that the car will no longer move straight after the manoeuvre which could result in crashes or accidents.

Thus, the curvature of the trajectory should be zero at both the start position and the end positions while turning or changing lanes when the vehicle is moving on a straight road.

2.4.4 Quartic Bézier curve based trajectory generation for autonomous vehicles with curvature and velocity constraints (2014)

Expanding on the prior solution for using fifth-order Bezier curves for lane changing [3], this paper [4] applied the principle of using Bezier curves of a high order for more

general trajectory generation. The model here was still predominantly rule-based and focused on lane-keeping and path following.

2.4.5 Bézier curve based trajectory planning for autonomous vehicles with collision avoidance (2020)

This paper's [28] solution builds upon Chen's solution to include collision avoidance within trajectory generation using fifth-order Bezier curves. The novel concept of using potential fields within the Frenet coordinate system is introduced (explained in section 4.3). The proposed method was found to be capable of generating collision-free trajectories for scenarios involving roads with two lanes and one moving and one stationary obstacle. However, this method was not extensively tested for more scenarios and factors such as the trajectory's computation time and smoothness were not evaluated for more extreme test cases.

This paper aims to explore and expand on this paper's work. Concepts such as potential field and path optimisation will be explored in depth and modifications will be made to this method to improve its performance for more varied scenarios. The performance of the algorithms developed in this project will then be thoroughly evaluated.

Chapter 3

Material and Methods

3.1 Bezier curves

This paper will utilize Bezier curves to derive the trajectory for vehicles. A Bezier curve is a smooth, parametric curve that is defined by a set of control points [19]. The reason Bezier curves seem to be ideal for path generation is due to the following properties:

- Control Points: A Bezier curve always starts at the first control point and ends at the last control point. The remaining control points determine the shape of the curve in between. This property ensures that the trajectory that is generated will always have the correct start and end positions.
- Convex Hull: The convex hull of a set of control points is the smallest convex polygon that contains all of the control points. The shape of the Bezier curve will always remain within the convex hull of its control points. This ensures that the generated trajectory will never go out of the bounds we set for it.
- Continuity: Bezier curves can be joined together to create a continuous curve. This means that bezier curves can be computed for small parts of the overall path and then joined together to create a continuous trajectory.
- Order of Curve: The order of a Bezier curve refers to the number of control points used to define the curve. By increasing the order of the Bezier curve, we can create more complex and intricate trajectories. Higher order Bezier curves can also produce more precise and smooth trajectories that better represent the natural motion of a vehicle.
- Curvature: Curvature is an important property to consider in generating trajectories since we would generally prefer smooth trajectories with low curvature. The curvature of a Bezier curve at a given point is the reciprocal of the oscillating circle. In figure 3.1 we can see that the curvature of curve C at point P can be found by taking the reciprocal of the radius of the blue circle. Thus, the curvature at this point is 1/r. For a straight line the radius of this circle would be infinite thus, its curvature would be zero.

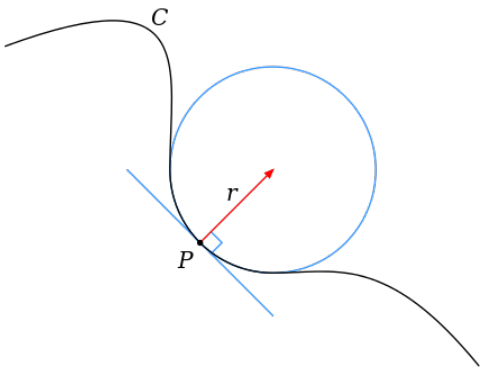


Figure 3.1: Curvature of Bezier curve C [8]

3.2 Software used

In order to accurately represent bezier curves in my project, I made use of the ndcurves library [21] which has in-built functions for representing and plotting bezier curves of an arbitrary order as well as the functionality for finding the derivative of the bezier curve. Additionally, I made use of the Scipy library [25] for solving optimisation problems.

Chapter 4

Implementation

4.1 Road model

For effective trajectory generation, the first factor to consider is the road. Given the variety of different roads- with some roads meandering in different ways- it is difficult to build a road model that can work with every road. Earlier papers by Zheng [28] and Chen [4] assume that the roads have well-defined lanes. However, this is not always the case, especially in developing countries such as India and Nigeria where roads tend to be not as well defined into separate lanes (as was shown in figure 1.1).

Our road model needs to be robust enough to address the complexity of such roads. We will start by addressing the complexity brought upon by curved roads using the Frenet coordinate system.

4.2 Frenet coordinate system

The Frenet coordinate system helps in dealing with curved roads since each point on the road is defined based on only two mutually perpendicular vectors. The first vector is called the tangent vector, which represents the direction of motion of the vehicle along the curve. The second vector is called the normal vector, which points in the direction perpendicular to the curve at that point. Together, these two vectors define the local coordinate system at each point on the curved road [14].

Using the Frenet coordinate system, we can decompose the motion of a vehicle on the road into two components: longitudinal given by the s-axis (in the direction of the tangent vector) and lateral given by the d-axis (in the direction of the normal vector). The longitudinal component determines the vehicle's speed, while the lateral component determines the vehicle's position relative to the center of the road.

Thus, any point on the road can simply be represented as an s,d coordinate. Decomposing the road in this manner allows us to apply the concept of potential field on roads.

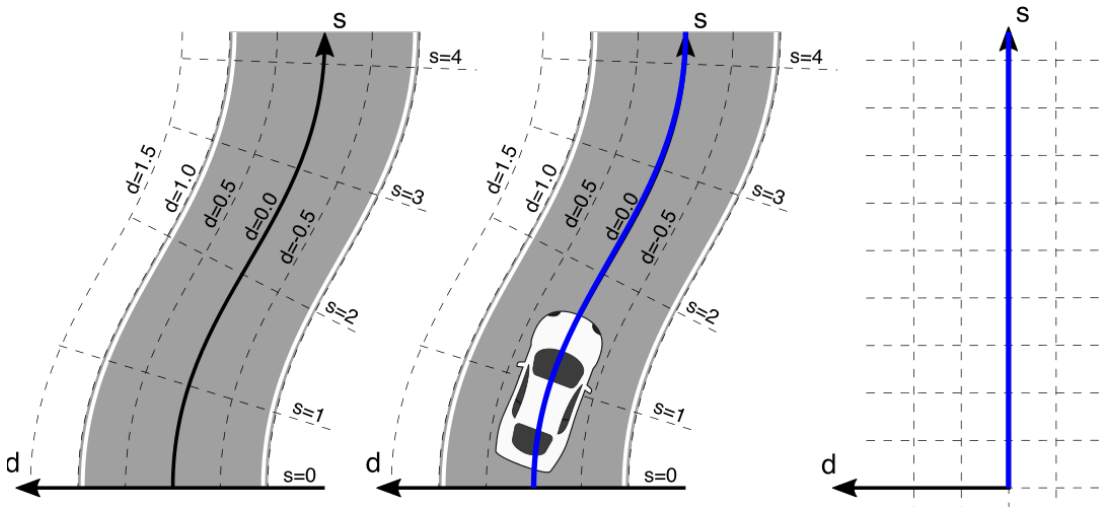


Figure 4.1: Translation of points on curved road in Frenet coordinate system [17]

4.2.1 Translating paths from Frenet coordinate system to Cartesian coordinates

Alternatively, we also need to define the translation from trajectories in the Frenet frame to Cartesian coordinates. For representing the road in Cartesian coordinates in this paper, I will be using the parametric equations defined by Zheng [28],

$$x(s) = a_3 s^3 + a_2 s^2 + a_1 s + a_0 \tag{4.1}$$

$$y(s) = b_3 s^3 + b_2 s^2 + b_1 s + b_0 \tag{4.2}$$

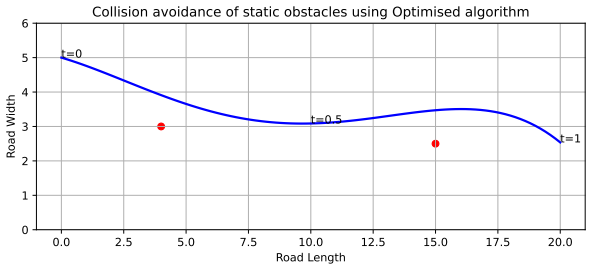
where a and b are constants that determine the specific shape of the curved road. Points along the width of the road can be found by calculating the heading angle θ, the angle of the direction in which the curve is moving [7], and plugging it in the following equations [28]:

$$\theta = \tan^{-1}\left(\frac{dy/ds}{dx/ds}\right) \tag{4.3}$$

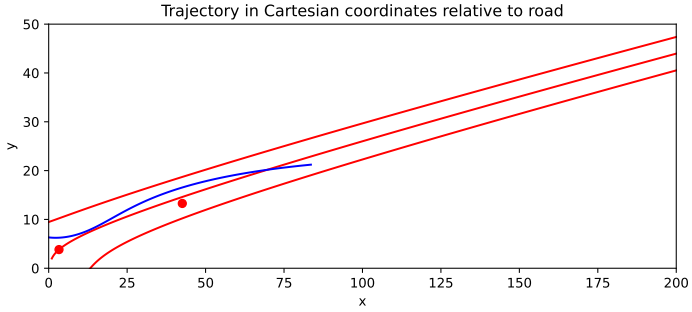
$$x_p(s) = x(s) - d\sin(\theta) \tag{4.4}$$

$$y_p(s) = y(s) + d\cos(\theta) \tag{4.5}$$

Using parametric equations allows us to simply plug in values of s and d from the Frenet space to get the corresponding curve in Cartesian coordinates. An example of this is shown in figure 4.2 where a trajectory generated in section 4.6.1 is translated to a road defined by parametric cubic equations. The curved road defined here takes a = (1, 3, 4 ,7) and b = (2, 4, 1, 1) as constants. It will act as the reference road for testing and the curvature of the trajectories will be calculated based on this reference road.



(a) Trajectory generated in Frenet coordinate space



(b) Trajectory translated to Cartesian coordinates on road

Figure 4.2: Translation of trajectory from Frenet coordinate space to Cartesian coordinates

In figure 4.2(b) the red lines are the edges and centre of the road and the red dots are obstacles.

4.3 Potential Field

The concept of potential field involves developing a probabilistic model where the possibility of a collision can be measured at any point on the road. Thus, the potential field allows us to find the danger at any point and we can then avoid dangerous regions when generating our trajectory.

4.3.1 Potential Field for roads

For developing the potential field for a road, I wanted a model that could account for roads that had lanes and those that did not.

In order to define the potential field I used the concept of “safe points” along the road. For instance, for a road with three lanes going in the same direction, the safest points on the road would be the centre lines of those three lanes. Conversely, the most unsafe points would be the edges of the road and the edges between two lanes.

In order to reflect this in the potential field, I started calculating the potential field for a point by calculating the distance from the point to the nearest safe line. The intuition being that the closer the point is to a safe point- the less danger there is. I then used the cosine function to calculate the danger at that point based on the ratio of the closest

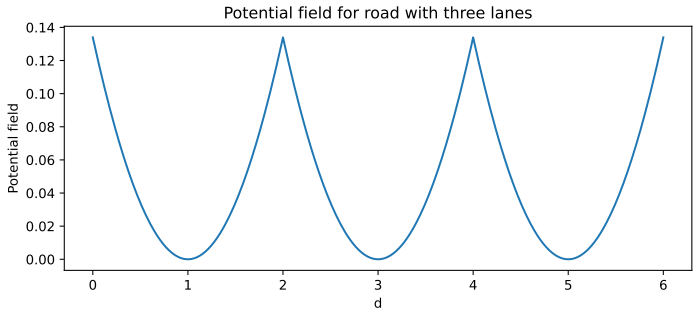


Figure 4.3: Potential field for road with three lanes

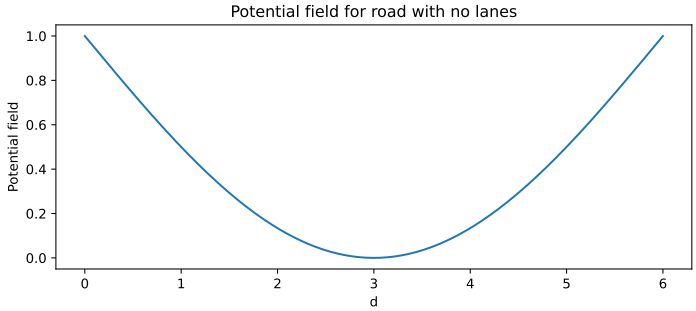


Figure 4.4: Potential field for road with no lanes

distance from a safe point to the width of the road. Thus, I developed the formula for potential field of a point on the road to be:

$$U_{road} = 0.5 \times \cos(\frac{\pi d}{L}) + 0.5 \tag{4.6}$$

Where d is the distance from the point to the nearest safe line and L is the width of the road.

In cases where there are no lanes- the safe points would simply be at the centre of the road. The potential field for a road with three lanes and a road with no lanes can be seen in figures 4.3 and 4.4 respectively.

4.3.2 Potential Field for vehicles and stationary obstacles

Along with the potential field of the road we also need to consider the danger that is posed by the other vehicles / obstacles on the road. In order to construct the potential field for these moving and stationary obstacles I used the formulation from Zheng [28] where we consider the position (s, d) and velocity (v) of these obstacles.

The first consideration is within what positional and velocity range should these obstacles be in order to be considered dangerous. We can disregard cases in which

- the obstacle is behind the ego vehicle and has less velocity than the ego vehicle

- the obstacle is ahead of the ego vehicle but has higher velocity than the ego vehicle

This leaves us with the following conditions:

$$v_e \geq v_i \wedge s \leq s_i \tag{4.7}$$

$$v_e < v_i \wedge s \leq s_i \tag{4.8}$$

Where v_e, s and v_i, s_i are the velocities and s position of the ego vehicle and the obstacle respectively.

For any other cases, the potential field would simply be 0.

For the potential field of such obstacles we are concerned with the distance between the ego vehicle and the obstacle as well as the velocity difference between them. Our potential field calculation will be determined by a distance function based on the distance between the ego vehicle and obstacle:

$$Dist = \left(\frac{(s-s_i)^2}{\delta_s^2} + \frac{(d-d_i)^2}{\delta_d^2}\right) \cdot (1/|v_e - v_i + \varepsilon|) \tag{4.9}$$

Where δ_s and δ_d are the standard deviations of s and d directions respectively and ε is an error term to avoid division by zero when the ego velocity and obstacle velocity are equal.

High velocity difference means that a collision is more likely if the ego vehicle and obstacle are moving in the same direction (in the cases we are considering). Thus, high velocity difference is reflected as reducing the distance between the objects in our distance function.

Negating the distance function and raising it to e ensures that when there is no distance between the objects the potential field is the highest i.e. $e^0 = 1$ whereas as the distance increases the potential field decreases exponentially. This gives the potential field of an obstacle on the road as:

$$U_{obstacle} = e^{-\left(\left(\frac{(s-s_i)^2}{\delta_s^2} + \frac{(d-d_i)^2}{\delta_d^2}\right) \cdot (1/|v_e - v_i + \varepsilon|)\right)} \tag{4.10}$$

In order to ensure that our generated potential field is appropriate I generated a few test scenarios to gauge the behaviour of the potential field. I started by defining my ego vehicle and giving it an arbitrary velocity of 10 units/s as well as deciding on a virtual road with no lateral component and s ranging from 0 to 10 units.

In the first instance, I created a vehicle with a velocity of 8 units/s and placed it at s=2. In this testing scenario, the vehicle remained stationary at this point but its velocity was still considered as 8 units/s as the ego vehicle moves from one end of the road to the other. I then calculated the potential field between this vehicle and the ego vehicle for 100 equally spaced points on the road and plotted the trend of the potential field.

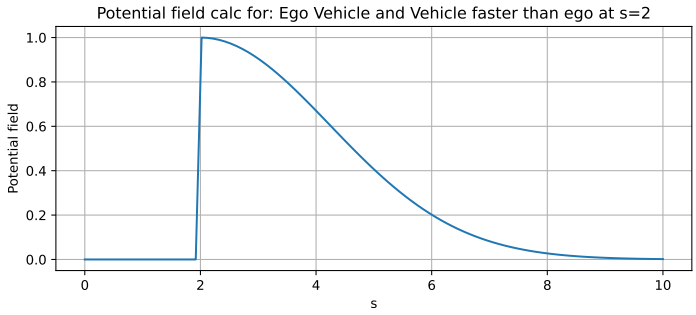


Figure 4.5: Potential field for ego vehicle and vehicle faster than ego vehicle at s=2

As seen in figure 4.5, the potential field is 0 while the ego vehicle is behind the faster vehicle since at this point the faster vehicle does not pose a threat. There is a steep increase as the vehicle approaches the faster vehicle at s=2. The decline in the potential field as the ego vehicle moves away is steady since even at s=6 the faster vehicle still poses a threat to the ego vehicle.

Alternatively, when testing with a vehicle that is given a velocity less than the ego vehicle (8 m/s)- there is less of a threat when the ego vehicle is moving ahead of this slower vehicle and the potential field quickly goes to 0 as shown in figure 4.6. In this case, there is more of a threat of crashing into the slower vehicle from behind shown by a consistent and moderate rise in the potential field as the ego vehicle approaches the slower vehicle from behind.

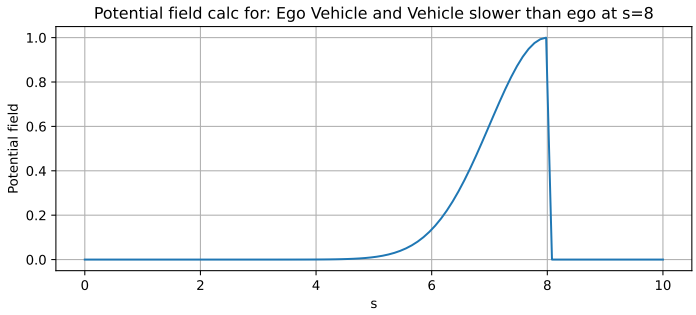


Figure 4.6: Potential field for ego vehicle and vehicle slower than ego vehicle at s=8

The potential field can be visualized in a more holistic way through the heatmap for which I consider the lateral offset d as well. From the heatmap in figure 4.7, we can see that the coordinates where the obstacles are present are regions of high intensity on the heatmap. The shape of these regions is also notable- the level of intensity of these diminishes as one moves farther away from the central location where the obstacle is positioned. Additionally, the regions of intensity are either behind or ahead of the obstacle based on its velocity relative to the ego vehicle.

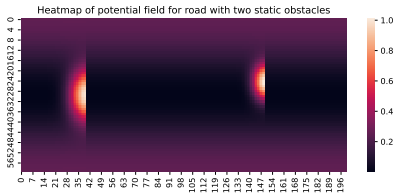


Figure 4.7: Heatmap of potential field for road with two static obstacles

4.3.3 Heuristic algorithm for trajectory generation based on potential field

Based on the intuition that as long as a car moves between a predefined range or angle its path will be smooth and continuous, I developed a simple heuristic algorithm using potential field.

The heuristic algorithm involves the following steps. First, define a “d range” - this will be the lateral range within which the algorithm will look for its next point. For instance, the d range can be $\pm$ 3 units on the d-axis from the ego vehicle’s current position. Since we would like to move at a constant velocity we can find the s value for the next point through the kinematic equation s = s + vt.

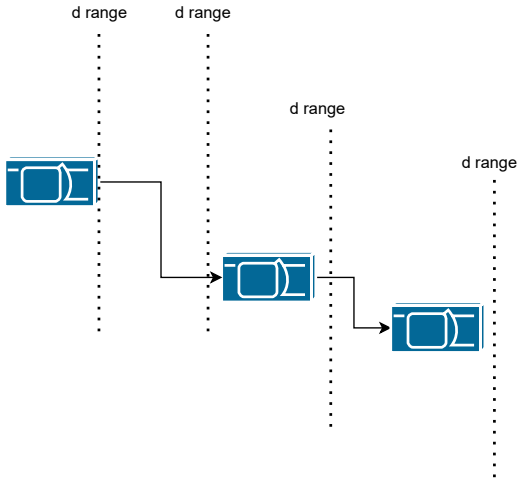


Figure 4.8: Diagram demonstrating concept of heuristic algorithm

Thus, the algorithm will look for the d value of the next point within the d range at this value of s. The algorithm will choose the value for d based on the point which has the minimum potential difference within the range. Once this point is found the ego vehicle will move to this new point and the algorithm will continue until the ego vehicle has reached the destination. These points can then simply be taken as the control points of a bezier curve and a smooth trajectory can be generated. It should be noted that the d range will be recomputed at every new point based on the d value at that point. Figure 4.8 explains this heuristic algorithm.

One advantage of the heuristic algorithm is its efficiency- the time complexity of the

algorithm can be taken as O(s * d) where s is the number of steps taken along the length of the road and d is the number of values in the d range. Since our priority is to generate a trajectory for the near future both these values would be quite low so the trajectory will be generated very quickly.

It can also be argued that since the d range will be quite a small window the vehicle doesn't have the option to make any sharp moves to the left or right. Thus, the generated path will be smooth and its curvature will be low. However, the small window of movement along the d-axis also makes it susceptible to a major problem associated with potential field which is getting stuck in a local minimum since the small window of movement inhibits the vehicle from moving to a more optimal path.

4.4 Curvature

While potential field helps in checking the possibility of collision for a given path, the path's curvature is also another important factor to consider while finding the optimal path. The curvature of the generated path directly affects the lateral acceleration of the vehicle. Vehicles with high lateral acceleration tend to be more unstable and difficult to control. Curvature is also directly linked to the smoothness of the path - thus it affects passenger safety and comfort. Sharp turns or sudden changes in direction categorised by high curvature can cause discomfort and even motion sickness. Thus we want to prioritise lower curvature. The curvature of a bezier curve r at a particular timestamp t is calculated with the following formula:

$$\kappa(t) = \frac{\|\vec{r}'(t) \times \vec{r}''(t)\|}{\|\vec{r}'(t)\|^3}$$

(4.11)

Prior work that involves measuring curvature of a path generated in Frenet space simply measures the curvature of the path itself [28]. However, this curvature may not reflect the actual curvature of the path once translated outside the Frenet space. In this paper when curvature is mentioned in the context of optimisation, I will be referring to the curvature of the curve when it is translated out of the Frenet space as mentioned in section 4.2.1.

Since there wasn't a function to get the curvature of a bezier curve within the ndcurves library- I used the curve derivate function in the library to develop my own function for getting curvature. The function will return the curvature at a particular timestamp t on the curve. This can be used within the optimisation to minimize the curvature of the entire curve.

4.5 Trajectory generation and optimisation

4.5.1 Sequential quadratic programming algorithm

In order to perform the optimisation for effectively generating smooth collision-free trajectories I used sequential quadratic programming. Sequential quadratic programming (SQP) is an iterative optimization algorithm that solves nonlinear constrained optimization problems [15]. SQP has been widely used in trajectory generation for road vehicles because it can handle nonlinear and nonconvex optimization problems, and it can efficiently handle a large number of constraints. In the context of trajectory generation for road vehicles, SQP can be used to find the optimal trajectory that satisfies certain constraints, such as the vehicle's kinematic constraints or the starting and end points for the trajectory, while minimizing a cost function that represents the desired behaviour of the vehicle.

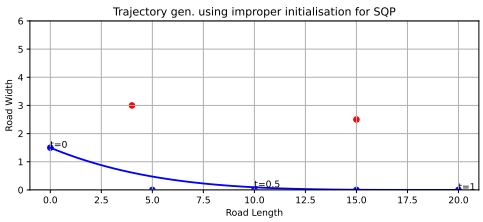
The SQP algorithm starts with an initial guess of the trajectory and iteratively improves it by solving a sequence of quadratic programming (QP) subproblems [15]. Each QP subproblem involves constructing a quadratic approximation of the cost function and the constraint functions around the current estimate of the trajectory, and solving for the optimal input that minimizes the approximation while satisfying the constraints. In our case, the optimal control input we are trying to find is the control points that form the optimal bezier curve trajectory. The solution of the QP subproblem is used to update the estimate of the trajectory, and the process is repeated until a convergence criterion is met.

4.5.2 Defining objective function

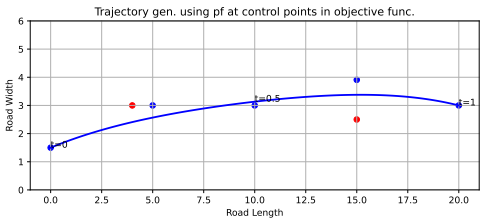
The first challenge with using SQP for finding the optimal path would be defining the objective function we would like to minimise. The objective function in the SQP algorithm is typically defined as a function of the decision variables that we want to optimize. In our case these decision variables would mainly be the potential field and the curvature.

To get a working solution I could build upon - I started by working only with the potential field. The first iteration of the objective function involved simply plugging in the heatmap I had generated for finding the potential field at various points on the road. I soon pivoted from this solution since I realised that I would need more precision to account for every point on the road which would require generating the potential field for more points which would in turn result in higher space and time complexity.

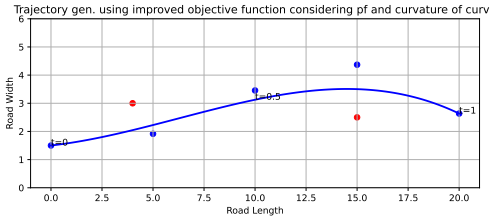
Instead, I decided to discretise the trajectory by considering only the control points of the bezier curve trajectory and calculating the potential field at these points. The objective function was then the summation of these potential field values. However, with this objective function, I was not getting the desired results.



(a) Trajectory generation using improper initialisation for SQP algorithm



(b) Trajectory generation using potential field at control points in objective function and constraints



(c) Trajectory generation using improved objective function considering potential field and curvature of the whole bezier curve

Figure 4.9: Improvements made to trajectory generation by altering constraints and objective function in SQP

4.5.3 Solving initialisation problem and adding constraints

One of the first problems I encountered while developing my algorithm was that the optimisation would tend to stick to the initial guess. In SQP, the starting point or initial guess for the algorithm can play an important role in the convergence of the algorithm. If the initial guess is too far from the true minimum, the algorithm may not converge to the correct solution.

My initial guess for the trajectory generation was moving straight along the edge of the road i.e. all the d values for the control points being close to 0 as shown in figure 4.9(a). This guess was likely to be a highly suboptimal choice since moving along the edge of the road would be very unsafe and result in a high potential field value. I eventually realised I was encountering an initialisation problem since despite making changes to my objective function the optimisation would still output the initial guess which outputted a cost of 1.125.

In order to correct this, I started by modifying the initial guess to be moving straight from the start point. I also wrote constraint functions for the lateral minimum and maximum- these were inequality constraints that ensured that the trajectory would never go to the edge of the road.

4.5.4 Problems with discretised objective function and improvements

Applying these constraints to the discretised objective function and altering the initialisation resulted in the generation of some more optimized trajectories. Figure 4.9(b) shows the trajectory generated for a test scenario using this objective function which outputted a cost of 0.196 (where dark blue points are the control points of the curve and red points are obstacles).

However, there were still drawbacks to this solution. The biggest drawback was that factoring in the potential field at control points did not necessarily result in a low potential field for the trajectory itself. On many occasions, control points can be far away from the curve that they are manipulating so even if the control point has a low potential field at a particular timestamp the points on the curve may still be in a region with a high potential field. Another drawback was that I could not always get a curvature calculation from only control points since they aren't always on the curve.

Thus, in order to account for the curvature and potential field of the complete curve and not just the control points, I further adjusted my objective function in python based on the optimisation problem defined by Zheng [28]:

$$minimise: J = \int_0^1 (w_\kappa \cdot \kappa^2(t) + w_u \cdot U_{total}(t))\, dt \tag{4.12}$$

I modified this problem to include weights for the curvature w_κ and potential field w_u. The inclusion of weights serves to indicate the higher importance of the potential field over curvature - since we would prefer to have a less smooth path rather than a less safe path. Thus, I started by setting w_κ as 0.1 and w_u as 0.9. Altering these weights to improve the performance of the algorithm will be the focus of Section 4.8.

I developed the objective function using my function for calculating curvature at a particular timestamp on a bezier curve and I also expanded on my prior function so that it would calculate the potential field for a particular timestamp on the curve. I used the quad integrate function within Scipy for defining the objective function as the integral of the summation of the curvature and potential field functions for the domain T is 0 to 1. I also set up an equality constraint to ensure that the start point of the bezier curve was at the start point of the vehicle.

Figure 4.9(b) shows the trajectory generated for the same test scenario using the new continuous objective function which outputted a cost of 0.047.

4.6 Trajectory generation

In this section, I will demonstrate the trajectory generation and collision avoidance that can be achieved with the algorithm.

4.6.1 Collision avoidance for static obstacles

Figure 4.10 shows a test case involving two static obstacles placed close to the centre of a road with a width of 6 units. In this instance the vehicle starts from a point close to the edge of the road- we can see that due to the higher risk of going off-track from this position, the vehicle is moving closer to the centre of the road. However, due to an obstacle being present at the centre of the road - the vehicle adopts a more gradual movement towards the centre of the road while maintaining a considerable distance from the obstacle. Due to the presence of another obstacle close to the centre further along the road, the ego vehicle maintains a slight distance from the centre and moves away slightly while approaching this obstacle thus decreasing the proximity between them.

Comparing this to the trajectory generated by the heuristic algorithm for the exact same test case, as seen in fig 4.10, the heuristic algorithm outputs a total cost of 0.085 as compared to the optimized algorithm which outputs a cost of 0.038. Similar to the optimized algorithm, the heuristic algorithm avoids both obstacles on the road while also having marginally lower curvature (0.0013) than the optimized algorithm (0.0014) although its higher cost can be explained by its proximity to the second obstacle.

4.6.2 Collision avoidance for moving obstacles

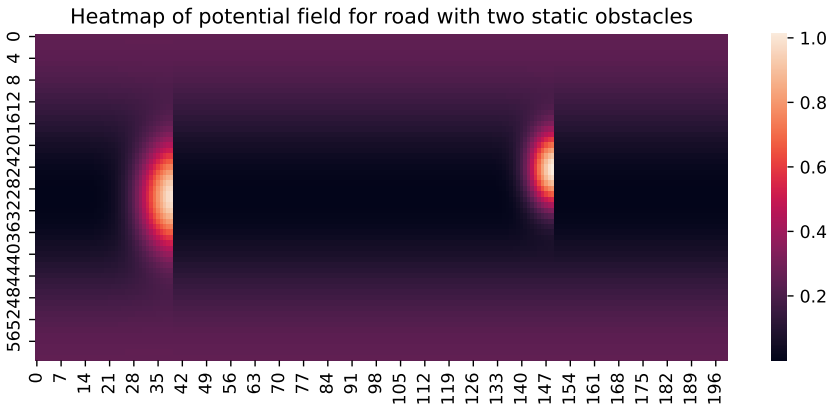
In real-world scenarios, other vehicles and pedestrians don't always move in a predictable manner. We cannot assume that every vehicle on the road prioritises safety to the same extent as the ego vehicle. Thus, the algorithm should be able to avoid moving obstacles that might move in random ways. In order to achieve this, I incorporated the sliding horizon control (SHC) method.

4.6.2.1 Sliding Horizon Control

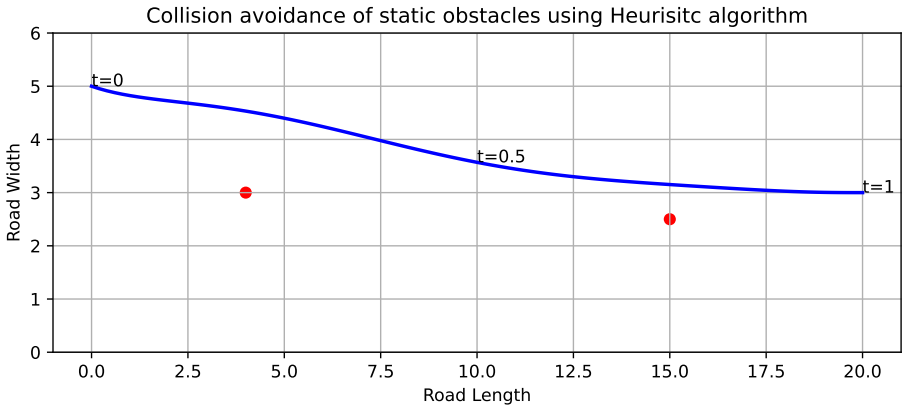
In SHC, the optimization problem is solved over a fixed time horizon, called the control horizon. At each control interval, only the first step of the optimal control sequence is applied, and the optimization problem is solved again using the updated state of the system. This process is repeated over time, resulting in a sliding horizon of predicted future behaviour. The advantage of SHC is that it can handle time-varying systems and disturbances. By only applying the first step of the optimal control sequence at each time step, SHC allows for flexibility in responding to changes in the system's behaviour.

In order to incorporate SHC, I extended my current implementation of the algorithm so that it would recompute a new trajectory at every timestamp t. For the purpose of this project, I set t to be 0.5. This meant that the algorithm would first compute the trajectory as a bezier curve. Then, the vehicle would move along the curve up until timestamp t. At this point, the algorithm would recompute another bezier curve to move ahead. The new computation would account for changes in the positions of obstacles within the time t. In this manner, the algorithm can adapt to changes in the environment such as the positions of obstacles changing or roads getting narrower.

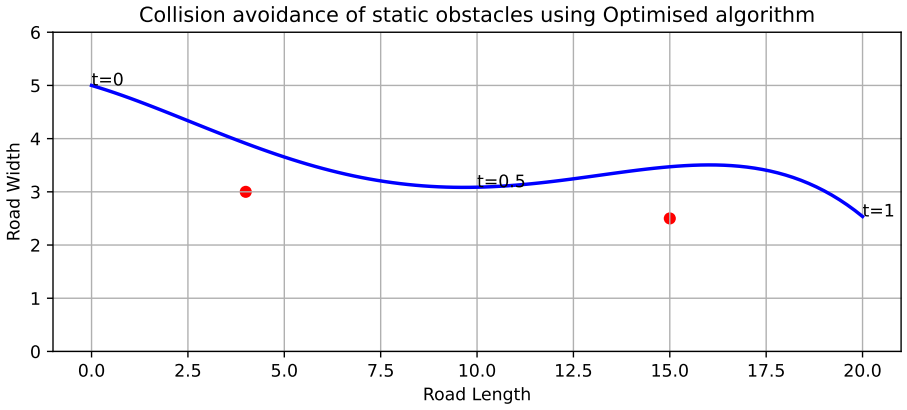
In order to make this process more efficient, the optimized path found at each iteration



(a) Heatmap of potential field for road with two static obstacles

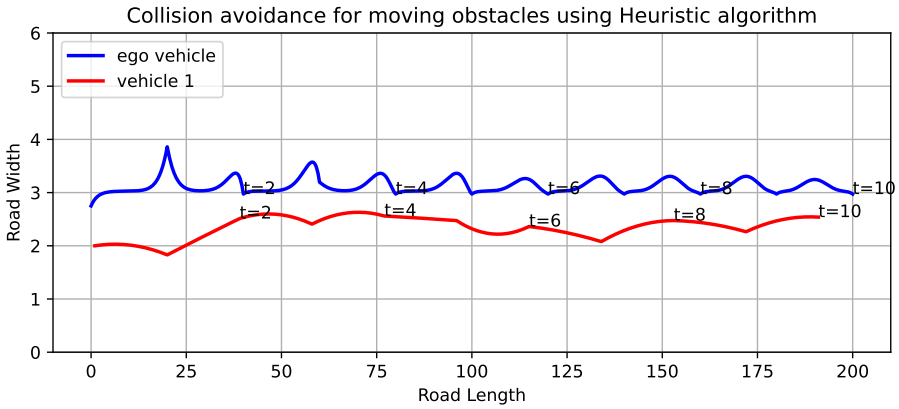


(b) Collision avoidance of static obstacles using Heurisitc algorithm

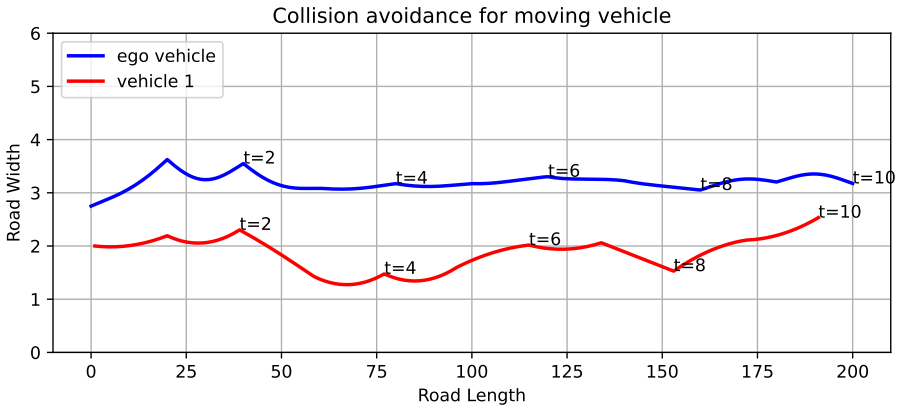


(c) Collision avoidance of static obstacles using SQP optimisation algorithm

Figure 4.10: Comparision of trajectory generation on road with two static obstacles for Heurisitic algorithm and optimized algorithm



(a) Collision avoidance for moving vehicle using Heuristic algorithm



(b) Collision avoidance for moving vehicle using SQP optimisation algorithm

Figure 4.11: Comparision of trajectory generation on road with a vehicle with random motion for Heurisitic algorithm and optimized algorithm

is used as the initial guess for the SQP algorithm in the next iteration. For instance, in the first iteration after moving to timestamp t- the algorithm will consider all the control points on the curve that haven't been visited yet and use these control points as the initial guess while finding the next curve. This ensures that the prior computation isn't wasted and makes the algorithm more efficient.

Figure 4.11 shows a test case involving a vehicle moving randomly traversing close to the ego vehicle. In order to generate the random motion of the moving vehicle, I set a window range within which the moving obstacle could move. The vehicle's next position is randomly selected from within this window iteratively. While the trajectories generated by this random generation are at times quite unusual it provides for effective and challenging test cases for the collision avoidance algorithms.

From figure 4.11(b) we can observe that using sliding horizon control allows the trajectory to adapt to the new position of the moving vehicle. At approximately s = 20 and s = 39, we can observe that the vehicle is supposedly moving towards the ego vehicle. In response to this, the ego vehicle quickly moves away resulting in a sharp turn

in the opposite direction. When the vehicle moves farther away from the ego vehicle, the trajectory becomes a lot more smooth and the ego vehicle moves straight- the total cost of the trajectory is 0.142.

Comparing this trajectory to the one generated by the heuristic algorithm in figure 4.11(a) shows the shortcomings of the heuristic algorithm. Since the heuristic algorithm doesn't consider curvature and simply moves to the point with the lowest potential field, the trajectory generated results in points of extremely high curvature when the distance between the vehicles reduces. While the trajectory does avoid collisions with the other vehicle, the resulting overall trajectory is practically impossible to follow for any road vehicle and has a cost of 0.733.

It should be noted that the optimized algorithm does also have points of high curvature. This can be improved by adjusting the weights for curvature and potential field within the objective function.

4.6.3 Trajectory generation with both vehicles running SQP optimisation algorithm

An important test case to consider is the scenario in which every vehicle on the road adopts the optimized algorithm for trajectory generation. In order, to simulate this I generated a test case with two vehicles, where the vehicles had velocities of 15 units/s and 19 units/s respectively.

In figure 4.12, we can observe that the ego vehicle is approaching the centre of the road at the start of the trajectory. When both vehicles are in close proximity they both react with sharp turns in opposite directions. The faster vehicle eventually overtakes the slower vehicle and moves to the centre of the road. Once there is a significant distance between the vehicles (at t=2) the slower vehicle also moves to the centre of the road and continues to move straight. The cost for the trajectories is 0.064 and 0.083.

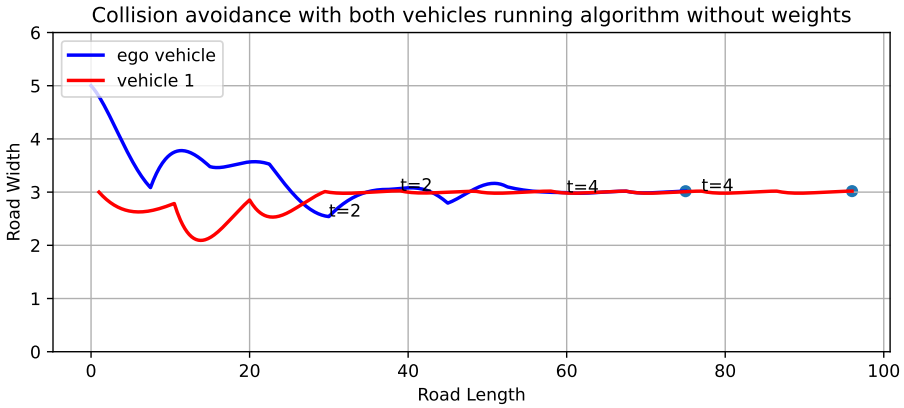


Figure 4.12: Trajectory generation with both vehicles running optimized algorithm

This test case brings forwards the main problems with the optimized algorithm. Firstly, there are only a finite amount of "safe zones" on the road. In this case, we are simulating a road with no lanes so the centre of the road is the region with the lowest potential

field. This means that any vehicle following the optimized algorithm would prefer to move to this region and it will switch to this region unless there is already a vehicle there. This results in crowding of the safe region. This can be seen in figure 4.12 as both vehicles eventually move to the centre at different times.

Additionally, there are also points of high curvature resulting from the vehicles trying to avoid each other. The high points of curvature can be explained by a heavy prioritisation of safety since currently, the weightage of the curvature to the potential field is 0.1: 0.9.

From the multiple points of high curvature along this path, it can be inferred that simply including curvature in our objective function is not enough to ensure that the resulting path is smooth and compliant with the vehicle model's limitations. Thus, it is essential to add some more constraints to the SQP optimisation according to the limitations of the vehicle model. It would also be beneficial to alter the weights using data instead of making an intuitive guess since this could result in smoother trajectories that avoid high points of curvature and also avoid collisions.

4.7 Ensuring paths satisfy restrictions of vehicle model

In order to ensure that the generated paths are traversable by the vehicle we need to take into account the vehicle's maximum steering angle, maximum steering rate, and maximum acceleration. Since we are generating trajectories assuming constant velocity, we do not need to check for acceleration in this case. To ensure that the vehicle's maximum steering angle is not exceeded when following a trajectory, I started by determining the maximum allowable curvature based on the maximum steering angle through the following formula [18]:

$$\kappa_{max} = \frac{\tan \delta_{max}}{W} \tag{4.13}$$

where κ_{max} is the maximum curvature, δ_{max} is the maximum steering angle, and W is the wheelbase. I then added a constraint within the SQP optimisation to ensure that the curvature at all points on the curve was below this value.

For finding the steering rate of the path we can use the following formula [18] and check that it does not exceed the maximum steering rate.

$$steering\ rate = \frac{\mathrm{d}\delta/\mathrm{d}t}{\mathrm{d}\psi/\mathrm{d}t} \tag{4.14}$$

where $\mathrm{d}\delta/\mathrm{d}t$ and $\mathrm{d}\psi/\mathrm{d}t$ are the gradients of the steering angle and heading angle with respect to time.

These checks can be added to our SQP optimisation as inequality constraints to ensure that the maximum steering angle and rate are not exceeded.

4.8 Adjusting weights for objective function in SQP optimisation

In order to adjust the weights in the SQP optimisation, I had to find the weights that provided for the best performance. Performance was measured by the number of points of proximity the trajectory had with obstacles (mentioned in section 5.1.1). This would also include the number of collisions.

My initial approach for finding the optimal weight was employing an optimisation method, where I would find the weight that minimised the number of points of proximitiy with obstacles. However, after visualising the data I realised that there is no direct relationship between the weight and the number of points of proximity/collision thus making it difficult to define a continuous objective function for optimisation.

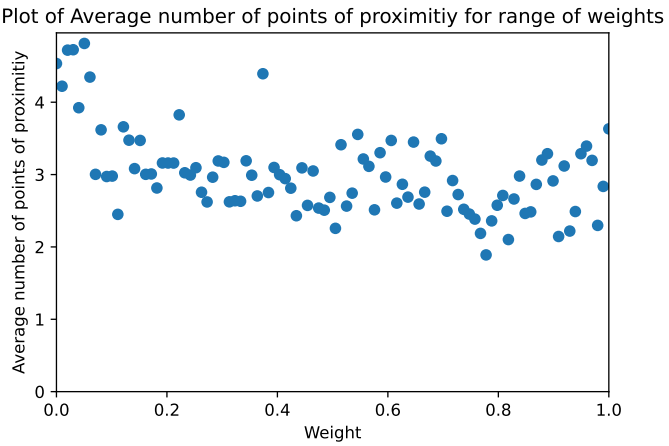


Figure 4.13: Plot of Average number of points of proximity for a range of weights w_u for the potential field in the objective function

Instead, I pivoted to a brute-force grid search.

- I defined a range of 100 equally spaced values from 0 to 1 as the weight for the potential field, w_u, in the objective function. The weight for curvature, w_κ, will be 1 - w_u.
- I then generated 25 test cases with 10 randomly generated obstacles as discussed in section 5.2.
- I generated a trajectory for each value in the range of weights for each test case and found the number of points of proximity between the trajectory and obstacles.
- I decided on the final weight as the weight that had the lowest average across 25 test cases which was 0.77 with an average of 1.889 points of proximity.

While grid search is inefficient, it only needs to be performed once and does not affect the run-time of the trajectory generation. That said, the grid search may not be able to find the absolute optimal weight since looking only at discrete data points does

not search for the optimum value in the input space exhaustively. However, it is still substantially better than the simplistic guess for the weight made earlier.

In figure 4.14, we can see the same test case involving two vehicles both running the optimized algorithm which was shown in figure 4.12. Now that the constraints for the vehicle model have been added and the weights for the objective function have been adjusted we can observe that the trajectory generated is a lot smoother and it can be easily traversed by a road vehicle. The overall curvature of the trajectories for the ego vehicle and vehicle 1 are 0.0003 and 0.0001 which is substantially lower than before (0.441 and 0.312). This contributes to a lower cost of 0.014 and 0.023

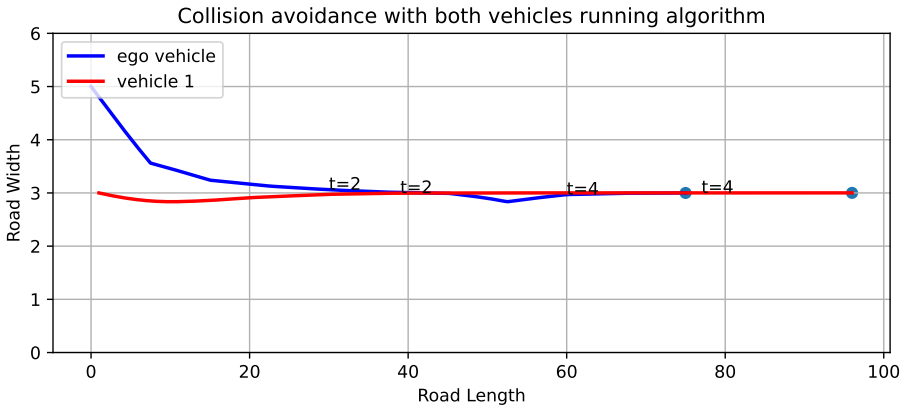


Figure 4.14: Trajectory generation with both vehicles running the optimized algorithm with constraints for vehicle model and adjusted weights

Chapter 5

Evaluation

In this chapter, I will discuss the metrics that were used to evaluate the performance of the trajectory generation. I will then critically analyse the performance of the algorithm based on these metrics and test cases and discuss the positive attributes and drawbacks of the algorithms.

5.1 Evaluation metrics for the performance of algorithms

5.1.1 Measuring the amount of collisions and proximity between vehicles

In order to evaluate the performance of the trajectory generation the most important criterion was checking for collisions. In order to check for collisions between two points, I started by defining the bounding sphere of radius 0.5 units for the vehicle. I then calculated the euclidean distance between the points and checked if the distance was below this collision radius. Thus, if the point was within the bounding sphere it would be considered a collision.

Another concern with trajectories is ensuring that vehicles are not in close proximity. While avoiding collisions is definitely the priority, vehicles being extremely close together can result in damage to the vehicles or other kinds of accidents. In order to check for such points of close proximity, I used the same functionality as for checking for collisions and simply increased the radius to 0.75.

The values for the collision radius and the proximity radius are can be altered based on a specific vehicle.

5.1.2 Computation time

Another important metric is the time taken to generate the trajectory. The computation time was measured by simply timing the execution of the functions for finding the trajectory in Python. In cases where Sliding Horizon Control was used, the computation time was measured for each sub trajectory and the maximum time taken was considered.

5.1.3 Highest point of Curvature

As mentioned before, the curvature of the trajectory is an important factor for safety and passenger comfort. However, using the overall curvature of the trajectory as a measure for the quality of the trajectory generated can be misleading. For example, if the trajectory includes a very sharp turn that is difficult to achieved within the limitations of the vehicle model, but the rest of the trajectory is predominantly straight and smooth, the overall curvature of the trajectory would still be relatively low. In this case, a suboptimal trajectory would be regarded as high-quality.

Additionally, the overall curvature of the trajectory is a factor that we are minimizing within the SQP optimisation. Thus, it does not make sense to use the overall curvature of the trajectory as a metric.

Thus, instead of measuring the overall curvature of the curve, I will record the highest point of curvature on the trajectory. Checking for the highest point of curvature would help in ensuring that trajectories don't have movements that are difficult to achieve for the road vehicle

5.2 Generation of test scenarios

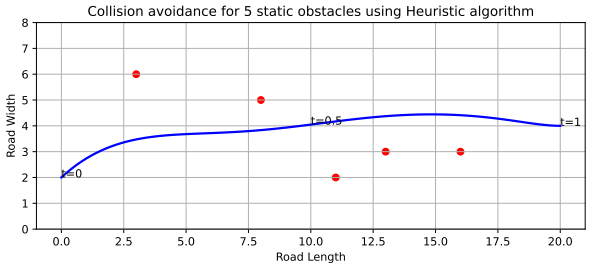
In order to evaluate the performance of the algorithms for a high number of varied test scenarios, I created the functionality to randomly generate roads of different widths within a range of 5 to 10 units based on the reference road shown in section 4.2 and place obstacles at random points on the road.

I decided on a few scenarios to test the method on- roads with 5, 10 and 20 static obstacles and roads with 1 and 3 moving obstacles. 25 tests were conducted for each scenario and the performance of the optimized and heuristic algorithms was measured for each test based on the aforementioned metrics.

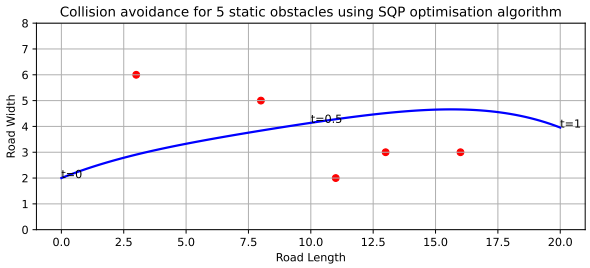
5.3 Analysis of trajectory generation performance for Static obstacles

For testing collision avoidance for static obstacles, I started by randomly generating roads of length 20 units with widths ranging from 5 to 10 units with five randomly placed obstacles along the road. Out of 25 trials, all trials were successful for the heuristic algorithm and the optimized algorithm. Successful trials refer to trials in which the algorithm generated a collision-free trajectory.

However, it should be noted that the heuristic algorithm is significantly faster than the optimized algorithm and its execution time is minimal and consistent. The average execution time for the optimized algorithm was 2.151 seconds and the maximum time taken was 3.931 seconds. On the other hand, the average execution time for the heuristic algorithm was only 0.001 seconds with minimal standard deviation. There also was not a considerable difference in the points of highest curvature for the two algorithms.

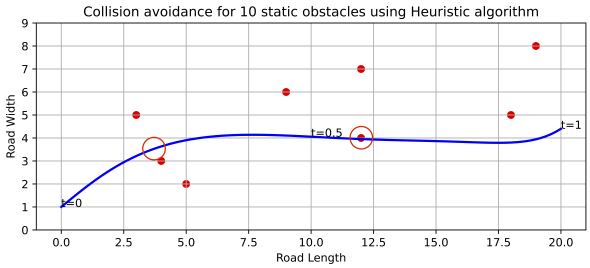


(a) Collision avoidance of 5 static obstacles using Heurisitc algorithm

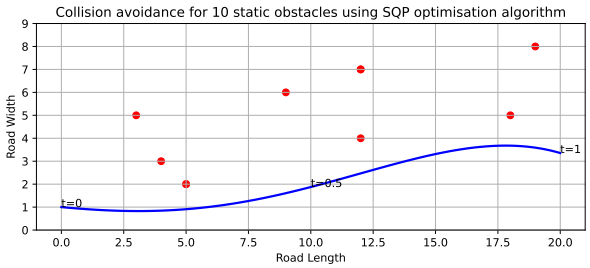


(b) Collision avoidance of 5 static obstacles using SQP optimisation algorithm

Figure 5.1: Comparision of trajectory generation on road with 5 static obstacles for Heurisitic algorithm and optimized algorithm



(a) Collision avoidance of 10 static obstacles using Heurisitc algorithm



(b) Collision avoidance of 10 static obstacles using SQP optimisation algorithm

Figure 5.2: Comparision of trajectory generation on road with 10 static obstacles for Heurisitic algorithm and optimized algorithm

Table 5.1: Performance statistics for optimized algorithm for 10 static obstacles

Statistic	Number of collisions	Number of points of proximity	Highest point of curvature	Computation Time (sec)
Mean	0.200	1.300	0.342×10^{-4}	3.174
std	0.116	1.580	0.046×10^{-3}	1.047
max	1.000	3.000	0.502×10^{-3}	4.762

Table 5.2: Performance statistics for heurisitic algorithm for 10 static obstacles

Statistic	Number of collisions	Number of points of proximity	Highest point of curvature	Computation time (sec)
Mean	1.500	7.400	0.001	0.002
std	1.509	5.080	0.001	0.001
max	4.000	15.000	0.502	0.003

Thus, for a low number of static obstacles on the road, it could be argued that the heuristic algorithm has better performance than the optimized algorithm since it generates predominantly collision-free trajectories in far less time. Figure 5.1 shows a comparison of the same test case for the heuristic and optimized algorithms.

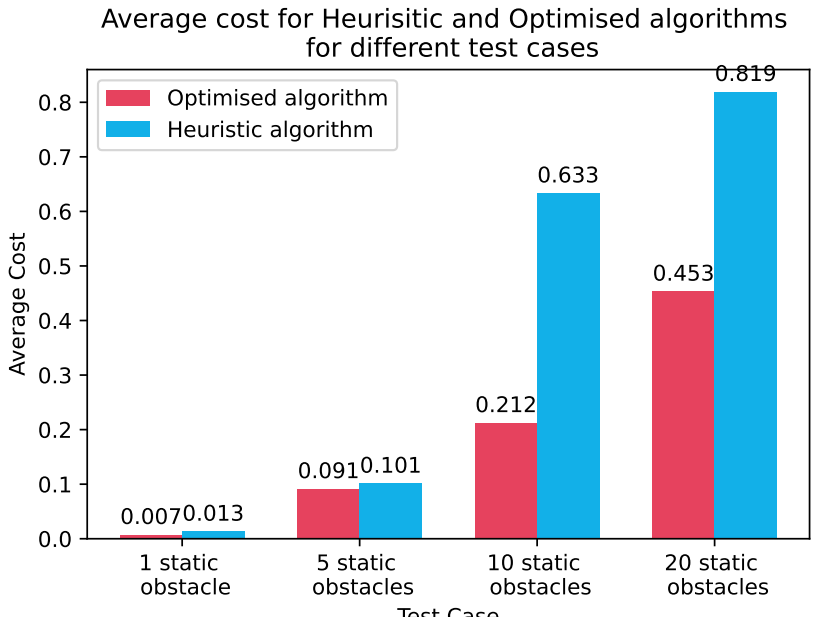


Figure 5.3: Average cost for Heuristic and optimized algorithms for different test cases

However, upon increasing the complexity of the problem by increasing the number of static obstacles on the road to ten, we can see the advantages of the optimized algorithm. While the average execution time for the optimized algorithm does increase to 3.174 seconds, the performance on other metrics such as the number of collisions and points of proximity remains consistent. Out of 25 trials, 24 are successful for the optimized algorithm whereas only 16 are successful for the heuristic. Figure 5.2 shows a comparison of the same test case for the heuristic and optimized algorithms through

Table 5.3: Performance statistics for optimized algorithm for 3 moving obstacles

Statistic	Number of collisions	Number of points of proximity	Highest point of curvature	Computation Time (sec)
Mean	0.600	6.300	0.272×10^{-2}	4.344
std	1.549	2.483	0.089×10^{-2}	2.047
max	3.000	10.000	0.608×10^{-2}	6.762

Table 5.4: Performance statistics for heurisitic algorithm for 3 moving obstacles

Statistic	Number of collisions	Number of points of proximity	Highest point of curvature	Computation Time (sec)
Mean	3.538	15.400	0.134	0.002
std	1.509	5.080	0.121	0.001
max	7.000	25.000	0.982	0.003

which we can see the advantages of the optimized algorithm. Due to its limited d range, the heuristic algorithm moves towards the centre of the road without accounting for collisions that can take place further along the road whereas the optimized algorithm evaluates the cost of that path and instead chooses to move along the edge of the road despite its high potential field since it avoids a collision.

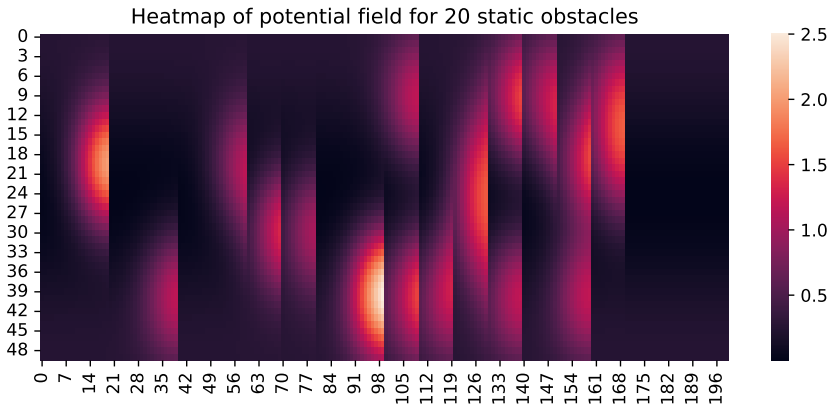
Finally, I stress-tested the performance of the algorithm by increasing the number of obstacles on the road to 20. In this scenario, finding a collision-free trajectory is nearly impossible in most test cases. However, the optimized algorithm was still successful in 13 out of 25 trials. Figure 5.4 shows an example of a trial with 20 static obstacles. The heatmap with the potential field for the 20 static obstacles demonstrates how unlikely it would be to generate a collision-free trajectory in this scenario. Even in such extreme cases, the optimized algorithm manages to find one of the best possible solutions. It should be noted that the optimisation algorithm had an average computation time only slightly higher than that for 10 obstacles (3.934 seconds) although the highest execution time recorded was 7.652 seconds.

5.4 Analysis of trajectory generation performance for Moving obstacles

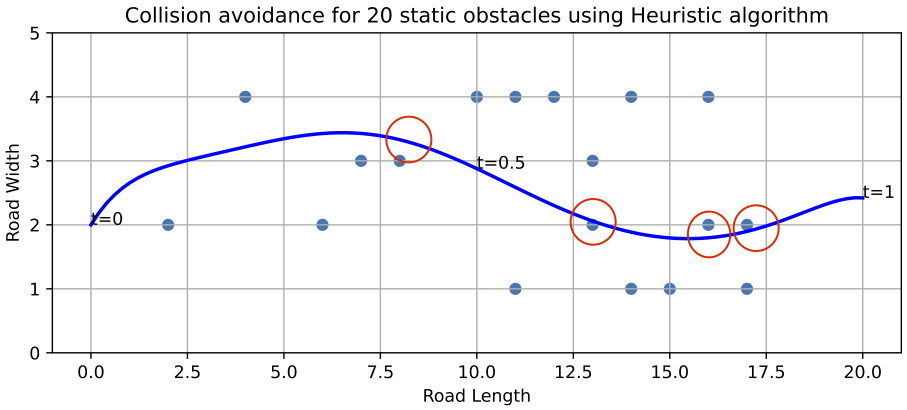
For testing collision avoidance for moving obstacles, I used the same model of randomly generating roads with an obstacle with random motion moving along the road.

Out of 25 trials, when testing for just one moving obstacle, 23 trials were successful for the optimized algorithm and 15 were successful for the heuristic algorithm.

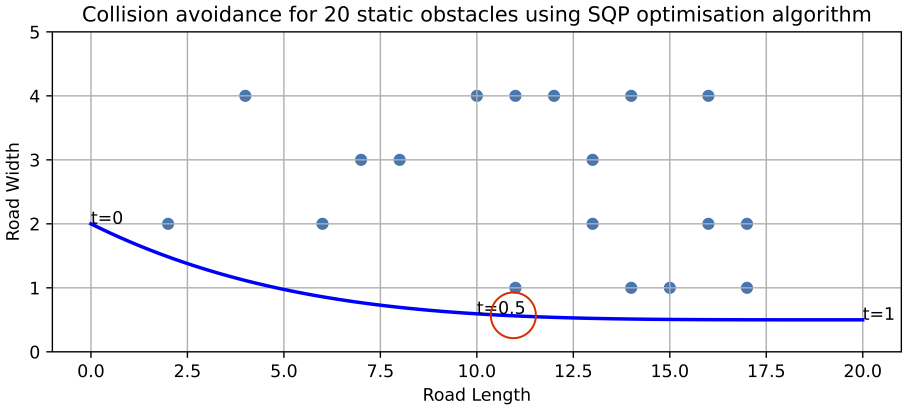
However, when the number of moving obstacles on the road increased to three, the success rate of both algorithms was affected drastically. For this scenario, I had the start point of the obstacles to be ahead of the ego vehicle and their velocity to be less than that of the ego. This meant that the ego vehicle would have to overtake the obstacles in this scenario.



(a) Heatmap of potential field for road with 20 static obstacles

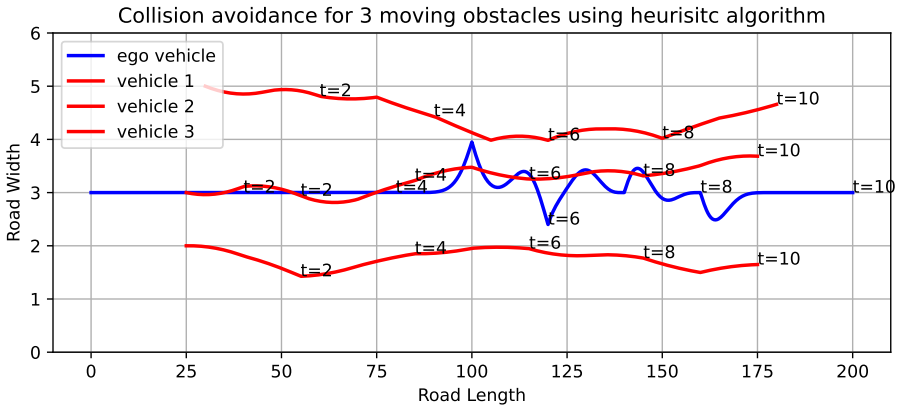


(b) Collision avoidance of 20 static obstacles using Heurisitc algorithm

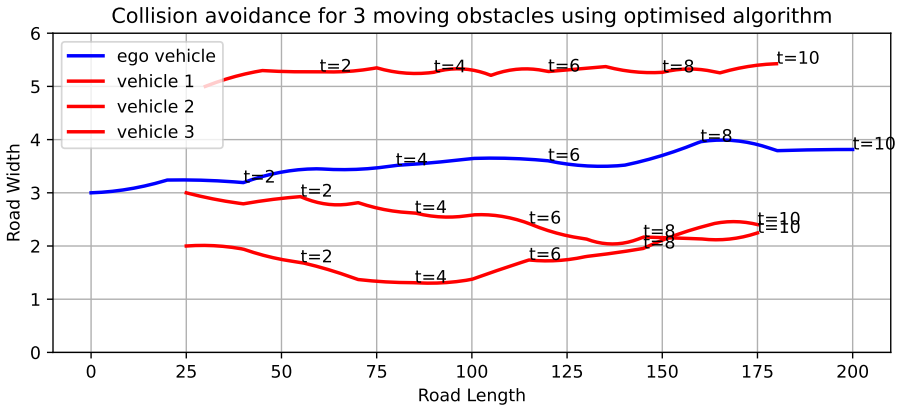


(c) Collision avoidance of 20 static obstacles using SQP optimisation algorithm

Figure 5.4: Comparision of trajectory generation on road with 20 static obstacles for Heurisitic algorithm and optimized algorithm



(a) Collision avoidance for 3 moving obstacles using Heuristic algorithm



(b) Collision avoidance for 3 moving obstacles using optimized algorithm

Figure 5.5: Comparision of trajectory generation on road with three vehicles with random motion for Heurisitic algorithm and optimized algorithm

Figure 5.5 shows the trajectories generated by both algorithms in such test cases. The irregular behaviour of the heuristic algorithm seen in 5.5(a) could be explained by the rapid changes in the potential field caused by the changes in the positions of the obstacles on the road. From this behaviour, we can conclude that the heuristic algorithm cannot be used in scenarios involving moving obstacles although its performance in scenarios involving static obstacles is noteworthy given its low execution time. On the other hand, the optimized algorithm was capable of finding a collision-free path in 20 out of 25 test cases. Figure 5.5(b) shows a test case in which the optimized algorithm finds a trajectory that avoids the moving obstacles by moving toward the safest region on the road. From the figure, we can see that once the ego vehicle overtakes the obstacles it begins to move straight ahead since it no longer has to react to the movement of the obstacles. Table 5.3 summarises the performance of the optimized algorithm in this scenario.

It should be noted that given the current success rate of the optimized algorithm in this scenario, further improvements are required. A contributing factor for the lower success rate could be that with moving vehicles there can be many situations where the ego

vehicle has no choice but to slow down and stop. Thus, the inclusion of rule-based methods such as ”stop the vehicle if there is no trajectory without an imminent collision” would be useful not just for this scenario but also for static obstacles.

5.5 Tradeoffs

5.5.1 Safety v/s Computation time

The first and most obvious tradeoff to consider in the trajectory generation algorithm is that of safety with computation time. While we would ideally always want the safest most optimal trajectory, computing this trajectory can require a considerable amount of time as seen with the optimized algorithm which took up to five seconds to compute a trajectory in certain cases.

It should be noted that computation time has an interesting relationship with safety. While more computation time could result in a safer trajectory- if the time taken for the generation of the next trajectory is too high the vehicle won't be able to decide its next move within the required time frame which can in turn result in accidents. Therefore, high computation time would in fact result in less safety.

Another drawback of the optimized algorithm was that its computation time was very inconsistent. For instance, the computation time for test cases involving 10 vehicles ranged from 1.322 seconds to 4.762 seconds. An inconsistent computation time is detrimental since it can lead to unpredictable behaviour of the vehicle. For example, if the algorithm takes longer to generate a trajectory in one instance, the vehicle may suddenly accelerate or brake to catch up with the generated trajectory, resulting in erratic behaviour.

Thus, the optimized algorithm would need to be more efficient and also have a more consistent computation time in order to be used in a real-world setting. The algorithm in this project was implemented using Python. While this informs us on the time complexity of the algorithm, the algorithm would also run a lot faster and more efficiently if it was written in a compiled language such as C++. However, it should be noted that writing the optimized algorithm in C++ will not necessarily reduce the inconsistency in the computation time of the algorithm since this inconsistency is likely to come from the varying complexity of the optimisation problem being solved.

5.5.2 Path smoothness v/s safety

In extreme cases where an obstacle is unexpectedly in close proximity to the ego vehicle, we would prefer to avoid the obstacle at the cost of passenger comfort. On the other hand, deprioritizing curvature can result in cases where the algorithm overreacts to a nearby obstacle and generates a path that involves a turn that is unnecessarily sharp.Thus, there is a complex relationship between these variables and aggressively prioritising one over the other can result in a major negative impact on the generated trajectories.

Chapter 6

Conclusions

6.1 Summary of work

Research in the field of trajectory generation for road vehicles using bezier curves has predominantly focused on the development and testing of streamlined scenarios that involve features like lane changing and path following. My goal for this project was to extend these methods for complex situations involving multiple obstacles and roads without lanes.

In this paper, I built upon Zheng's method [28] for collision-free trajectory generation based on potential field and fifth-order bezier curves. Zhengs method for trajectory generation relied on the existence of at least two lanes on the road so that the ego vehicle could simply switch lanes when avoiding obstacles. It was demonstrated that this method can only evade a single obstacle at any given time.

In this paper, I designed the potential field for roads to include cases where the road did not have lanes. I then made use of SQM to find the optimal trajectory. Unlike the prior method which gave curvature and potential field equal importance, I included weights in the objective function for curvature and potential field in SQM and then used grid search to alter the weights based on the weights that had the best performance. I demonstrated that this method was capable of generating trajectories that avoided multiple static obstacles simultaneously.

In order to deal with moving obstacles, I incorporated sliding horizon control in the method so that the trajectory could react to changes in its environment. The trajectory generation method was then tested for a variety of scenarios including roads with multiple stationary obstacles, multiple moving obstacles as well as two vehicles running the trajectory generation algorithm at the same time. I evaluated the performance of trajectories based on the number of points of collisions and proximity with obstacles as well as the highest point of curvature on the trajectory and the time taken for generation. While I found that the optimised algorithm was capable of avoiding collisions with one moving obstacle on the road, its performance suffered when multiple obstacles on the road were moving unpredictably.

Through these tests, I identified the shortcomings of the algorithm and attempted to

make the relevant changes to improve it. For instance, testing on two vehicles running the trajectory generation at the same time showed that the trajectory could exceed the maximum steering angle of the vehicle in extreme cases while avoiding collisions. Thus, I set up tighter constraints for the vehicle model in the SQP optimisation.

Thus, in this paper, I developed a trajectory generation method for road vehicles that is capable of generating collision-free trajectories for various challenging situations. However, considerable future work and testing would be required in order to incorporate this method in real-world systems.

6.2 Limitations and Future work

6.2.1 Improvements to method

In this paper, I used a very basic model for representing the vehicles and obstacles wherein a vehicle and its position were defined by a point in the coordinate space and a collision sphere. In order to make the method ready for real-world usage it would be necessary to incorporate a more complex vehicle model which would account for the vehicle's specific dimensions, shape, weight, and tire type.

Additionally, in this paper, the trajectories generated assumed that the velocity of the vehicle would be constant throughout the trajectory. While aiming for a constant velocity is ideal and ensures passenger comfort in the vehicle, it is a myopic and limiting assumption in the context of trajectory generation. An improved method should allow the vehicle to change its velocity in response to changes in the environment. Which would allow the vehicle to slow down when approaching an obstacle or increase its velocity when overtaking a slower obstacle. The ability to change the ego vehicle's velocity will definitely make the method more versatile and capable of dealing with a wider variety of situations.

While the priority for this paper was ensuring passenger safety and comfort and using optimisiation to find the trajectory that would be the safest, allowing for the velocity to change during the course of the trajectory means we can also optimize other things such as minimising the travel time or the fuel consumed.

Currently the method assumes complete perception of the road and the positions of obstacles. However, in practice this perception will be based on data collected from sensors on the vehicle. It would be unreasonable to assume that the sensors would be able to gauge the position of obstacles that are not in its direct proximity. Thus, the method should also be capable of working in scenarios where there is partial perception of the environment. While the incorporation of sliding horizon control does help with this problem, further improvements could involve predicting the movements of obstacles using machine learning [24] instead of only relying on perception.

Another major shortcoming of the proposed method in testing was the computation time. The proposed method would need to be more efficient with a consistent computation time in order to be used in real-world settings. One way of improving the computation time would be rewriting the algorithm in a compiled language such as C or C++ and

using a more capable computer.

6.2.2 Improvements to testing

The trajectory generation methods demonstrated in the paper have been tested within the Frenet coordinate space. The translation out of Frenet space and the measurement of curvature of the path in Cartesian frame was only done based on one predetermined curved reference road. However, in real-world scenarios there are a variety of different road shapes and the method would need to be tested for such roads with varying curvatures. Modifications would definitely be needed to be made to the current methods for the trajectory generation to work for roads with high curvature such as U-turns or roundabouts. These modifications would most likely employ rule-based methods of trajectory generation that would describe the expected behaviour of vehicles at structures such as roundabouts. It would then require considerable testing involving a large data set of roads and their shape to effectively ascertain that the trajectory generation works effectively outside of the Frenet coordinate space.

Most tests in the project involved the other obstacles moving ahead or along the same start point as the ego vehicle. Thus, it would also be essential to have more test cases wherein obstacles are approaching the ego vehicle at a higher velocity from behind it. Functionality could also be built into the method to slow down the ego vehicle in order to let faster vehicles overtake it easily.

In real-life scenarios roads have a combination of stationary and moving obstacles- additional tests could also be set up to include scenarios such as these. Additionally, a higher range of velocity for the ego vehicle and the obstacles should also be considered since most tests in the project had the ego velocity range from 15 to 20 units/s.

While the optimized algorithm was tested against the heuristic algorithm during testing, it would be more beneficial to compare the algorithm to other state-of-the-art methods for trajectory generation [11] to effectively evaluate the performance of the optimized algorithm against a capable method.

Overall, while the proposed trajectory generation method showed some promising results in testing, considerable further improvements and refinements are necessary to make it viable for real-world implementation in autonomous driving systems.

Bibliography

[1] Berkley Alexandre M. Bayen. Eliminating traffic jams with self-driving cars. https://ce.berkeley.edu/news/2537.

[2] Gus Alexiou. How passengers with disabilities can drive the autonomous vehicle revolution, Apr 2021.

[3] Il Bae, Jaeyoung Moon, Hyunbin Park, Jin Hyo Kim, and Shiho Kim. Path generation and tracking based on a bezier curve for a steering rate controller of autonomous vehicles. In *16th International IEEE conference on intelligent transportation systems (ITSC 2013)*, pages 436–441. IEEE, 2013.

[4] Cheng Chen, Yuqing He, Chunguang Bu, Jianda Han, and Xuebo Zhang. Quartic bézier curve based trajectory generation for autonomous vehicles with curvature and velocity constraints. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6108–6113. IEEE, 2014.

[5] Jiajia Chen, Pan Zhao, Tao Mei, and Huawei Liang. Lane change path planning based on piecewise bezier curve for autonomous vehicle. In *Proceedings of 2013 IEEE International Conference on Vehicular Electronics and Safety*, pages 17–22. IEEE, 2013.

[6] Ji-wung Choi, Renwick E Curry, and Gabriel Hugh Elkaim. Continuous curvature path generation based on bézier curves for autonomous vehicles. *IAENG International Journal of Applied Mathematics*, 40(2), 2010.

[7] Manfredo P. Do Carmo. *Differential Geometry of Curves and Surfaces*. Dover Publications, Mineola, NY, 1st edition, 2016.

[8] Manfredo P. Do Carmo. Cagd/bézier curves, 2018.

[9] Ferenc Hegedüs, Tamás Bécsi, Szilárd Aradi, and Péter Gápár. Model based trajectory planning for highly automated road vehicles. *IFAC-PapersOnLine*, 50(1):6958–6964, 2017.

[10] John Horst and Anthony Barbera. Trajectory generation for an on-road autonomous vehicle. In *Unmanned systems technology VIII*, volume 6230, pages 866–877. SPIE, 2006.

[11] Christos Katrakazas, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-

the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60:416–442, 2015.

[12] Joe Loncraine. Street life in lagos, 2011.

[13] NHTSA Media. Usdot releases 2016 fatal traffic crash data, Oct 2017.

[14] Prof David Murray. A1 vector algebra and calculus, 2015.

[15] Jorge Nocedal and Stephen J Wright. Sequential quadratic programming. *Numerical optimization*, 2:503–526, 1999.

[16] World Health Organization. Sdg target 3.6 road traffic injuries. https://www.who.int/data/gho/data/themes/topics/sdg-target-3_6 $-$ *road* $-$ $traffic-injuries$.

[17] Franz Pucher. Trajectory planning in the frenet space, Aug 2018.

[18] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[19] Peter Shirley and Steve Marschner. *Fundamentals of Computer Graphics*. CRC Press, Boca Raton, FL, 3rd edition, 2009.

[20] Richa Taneja. Hyderabad ranks third among most sound polluted cities, 2018.

[21] Steve Tonneau, Jason Chemin, Pierre Fernbach, and Guilhem Saurel. ndcurves, 2013.

[22] Department for Transport. Self-driving revolution to boost economy and improve road safety, Aug 2022.

[23] Bill Triggs. Motion planning for nonholonomic vehicles: An introduction. 1993.

[24] Jaskaran Virdi. *Using deep learning to predict obstacle trajectories for collision avoidance in autonomous vehicles*. University of California, San Diego, 2018.

[25] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[26] Patrick Winn. Will driverless cars work outside the united states?, Sep 2017.

[27] Xiaoxue Zhang, Jun Ma, Zilong Cheng, Sunan Huang, Shuzhi Sam Ge, and Tong Heng Lee. Trajectory generation by chance-constrained nonlinear mpc with probabilistic prediction. *IEEE Transactions on Cybernetics*, 51(7):3616–3629, 2020.

[28] Ling Zheng, Pengyun Zeng, Wei Yang, Yinong Li, and Zhenfei Zhan. Bézier curve-based trajectory planning for autonomous vehicles with collision avoidance. *IET Intelligent Transport Systems*, 14(13):1882–1891, 2020.