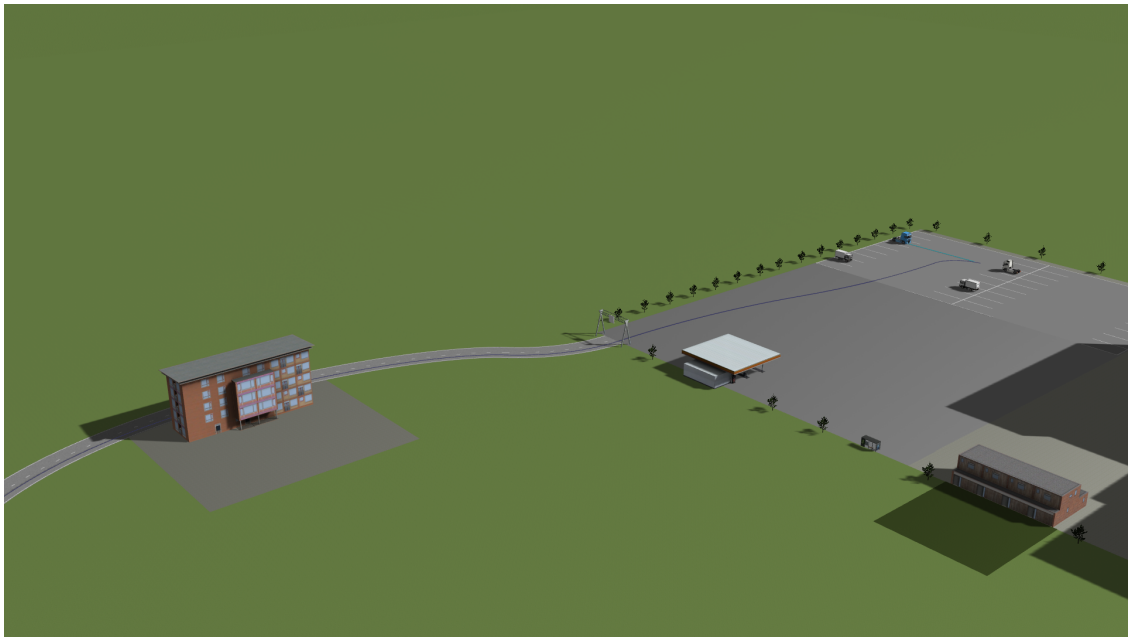

Autonomous Vehicle Navigation and Parking

Implementation of Path Planning and Trajectory Generation

Master's thesis in Systems, Control and Mechatronics

ANTHONY LABBÉ
JAYESH BHATT

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018:EX085

Autonomous Vehicle Navigation and Parking

Implementation of Path Planning and Trajectory Generation

ANTHONY LABBÉ
JAYESH BHATT

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2018

Autonomous Vehicle Navigation and Parking
Implementation of Path Planning and Trajectory Generation
ANTHONY LABBÉ
JAYESH BHATT

Supervisor: Torgrim Brochmann, AVL Gothenburg
Supervisor: Johan Janneson, AVL Regensburg
Examiner: Nikolce Murgovski, Department of Electrical Engineering

Master's Thesis 2018:EX085
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Example of trajectory generation in a valet-parking scenario.

Typeset in LaTeX
Gothenburg, Sweden 2018

Autonomous Car and Truck Navigation and Parking
Implementation of Path Planning and Trajectory Generation
ANTHONY LABBÉ
JAYESH BHATT
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Path planning and trajectory generation are in an important layer of an autonomous vehicle system. This layer is used to generate feasible trajectories for a vehicle so that it can navigate comfortably, safely and according to the road rules. A trajectory is a continuous sequence of states that forms a path and includes a suitable velocity. The trajectories are computed based on the information from a perception and an environment model. In this thesis, the aim is to achieve a trajectory generation layer such that an autonomous valet parking can be completed in a simulation environment. This can be solved by various methods. To accomplish the task, two different methods are used. One method is to used to navigate the vehicle on a road to a parking lot, the other method is used to park the vehicle once it has reached the parking lot. Subsequent to this, static and dynamic obstacles are to be avoided throughout the navigation. Road boundaries and other necessities have to be respected.
The proposed algorithms are a Conformal Spatiotemporal State Lattice algorithm and a Hybrid-A* algorithm. In the State Lattice, cubic polynomials are generated to given waypoints along a road and numerous velocity profiles are applied. These are the trajectories from which a suitable one is chosen based on a defined cost function. The Hybrid-A* provides waypoints to the goal based on a shortest-path search method. Continuous (Reeds-Shepp) curves are generated to the waypoints.
The results showed successful trajectory generation according to the aim. The State Lattice method strongly obeyed the defined constraints but were dependent on well placed waypoints. The Hybrid-A* method completed the task but was discovered to be narrowed to specific scenarios.
Improvement of both algorithms are needed to make them more generic. Optimisation of the cost function, the computational time and the waypoint definition are factors to be investigated for improvements in the former method. The latter method revealed to be less reliable in terms of safety and accuracy. This is because the method is highly dependent on constructively defined prior maps. Additionally, a good controller is needed to proceed with further optimisation of both methods.

Keywords: Path planning, Trajectory generation, Hybrid-A*, Reeds-Shepp's model, Algorithms, Polynomial method, State Lattice planner, Autonomous drive, Parking, Navigation

Acknowledgements

We would like to thank AVL Gothenburg and AVL Regensburg for making this thesis possible. It has been a challenging yet interesting journey to be exposed to a problem that is evolving. We extend special gratitude to our supervisors, Torgrim Brochmann and Johan Jannesson, for all the support and for consistently monitoring our work.
We would also like to extend special gratitude to our examiner, Professor Nikolce Murgovski of Chalmers University of Technology, for the competent advice. Every meeting resulted in interesting discussions which enlightened us throughout the thesis.
We are very grateful to Bernardo Henriques and Panos Tsilivis at AVL Regensburg, for the technical support. They guided us towards the goal throughout the project, by enlightening us with their supportive knowledge. We acknowledge Panos for being helpful and welcoming at every situation.
Finally, we would like to thank all the colleagues at AVL Gothenburg for giving us a comfortable environment and everyone at AVL Regensburg for the inspiration. We would also like to thank our family and friends for their support.

Anthony Labbé, Jayesh Bhatt, Gothenburg, August 2018

Contents

List of Figures **xi**

List of Tables **xix**

1 Introduction **1**
1.1 Background . . . 1
1.2 Aim . . . 2
1.3 Scope and Limitations . . . 3
1.4 Thesis Outline . . . 4

2 Theory and Related work **5**
2.1 Theoretical Preliminaries and Definitions . . . 5
2.1.1 Path Planning . . . 6
2.1.2 Trajectory Generation . . . 6
2.2 Mathematical Preliminaries . . . 6
2.2.1 Composite Simpson's Rule . . . 6
2.2.2 Newton-Raphson Method . . . 7
2.3 Vehicle Model . . . 7
2.4 Planning Algorithms . . . 9
2.4.1 Working of A* Algorithm . . . 10
2.4.2 Explanation of Reeds-Shepp Algorithm . . . 11
2.4.2.1 Reeds-Shepp's method . . . 13
2.5 State lattice . . . 18
2.6 Hybrid-A* . . . 19

3 Method of Path Planning and Trajectory Generation **21**
3.1 Structured Road Method . . . 21
3.1.1 Architecture of Online Structured Road Method . . . 23
3.1.1.1 Waypoint Processing . . . 24
3.1.1.2 Path Planner . . . 25
3.1.1.3 Newton-Raphson Solution . . . 28
3.1.1.4 Speed Generation . . . 30
3.1.1.5 Cost Functions . . . 31
3.1.1.6 Trajectory Application . . . 33
3.1.2 Real-Time Configuration . . . 34
3.2 Unstructured Environment Method . . . 35
3.2.1 Architecture of Unstructured Environment Method . . . 35

3.2.2 Inputs and Outputs . . . 37
3.2.3 Hybrid A-star . . . 38
3.2.4 Reeds-Shepp Curve Generation . . . 40
3.2.5 Acceleration Profile and Wheel Angle . . . 43

4 Results **45**
4.1 Offline Structured Road Method . . . 45
4.1.1 Cost Function Configuration . . . 46
4.1.2 Plot Results . . . 47
4.2 Offline Unstructured Road Method . . . 51
4.2.1 Parameter Configuration . . . 51
4.2.2 Test Scenario 1 . . . 51
4.2.3 Test Scenario 2 . . . 53
4.2.4 Test Scenario 3 . . . 53
4.3 Simulation Results . . . 55
4.3.1 Online Structured Road Method . . . 56
4.3.1.1 Motion Results . . . 57
4.3.1.2 Path Generations . . . 59
4.3.1.3 Trajectory Generation . . . 61
4.3.2 Online Unstructured Road Method . . . 63

5 Discussion **67**
5.1 Structured Environment Method . . . 67
5.1.1 Offline . . . 67
5.1.2 Online . . . 68
5.2 Unstructured Environment Method . . . 69
5.3 Merging Algorithms . . . 70

6 Conclusion **73**
6.1 Future Work . . . 74

Bibliography **77**

A Additional Plots **I**

B Common Algorithms **V**
B.1 Path Finding . . . V
B.2 Path Generation . . . VI

List of Figures

1.1 Example of the architecture of an autonomous system. 2

2.1 Kinematic bicycle model with no-slip assumption, as seen in [2]. L is the length between the front and rear axle. δ is the front wheel angle, θ represents the heading angle of the vehicle, *i.e.* $g_f - g_r$. g_r and g_f indicate the ground contact point of the rear and front wheel, respectively. $\hat{e}_x$ and $\hat{e}_y$ are unit vectors in (x, y) coordinates. 8

2.2 Illustration of A* path, showing all possible neighbouring nodes to the current node. Here, the cyan circle at $(5, 5)$ is the initial node. The green circle at $(15, 15)$ is the current node and the red circle is the goal node, the magenta circles represent the neighbours surrounding the current node. The X and Y axes represent the dimension of the map. 10

2.3 Zoomed view of the A* path calculating the next step. This figure shows the chosen node among the neighbour nodes, from the current node. The green circle is the current node and the red circle is the goal node, the dark green circle is the chosen node added to the path, towards the goal node. The X and Y axes represent the dimension of the map. 11

2.4 The blue line illustrates a path by A* algorithm, showing the shortest path between two points. The goal is depicted to be inside the parking place, which is represented by the red circle. An A* path is generated from the initial position, represented by the green circle, to the goal. . 12

2.5 Illustration of the A* path, showing that the path also avoids obstacles, placed in-between the environment (the black boxes). 12

2.6 Illustration of Reeds-Shepp curve, showing the $L^-S^-L^-$ curve which is generated by the *CSC word* of the algorithm. The curve, depicted by the blue path, is generated with an initial angle of 90° and goal angle of 270°. 16

2.7 Illustration of Reeds-Shepp curve, showing the $L^+S^+L^+$ curve which is generated by the *CSC word* of the algorithm. The curve, depicted by the blue path, is generated with an initial angle of 0° and goal angle of 90°. 16

2.8 Illustration of Reeds-Shepp curve, showing the $R^-L^-R^+$ curve which is generated by the *CCC word* of the algorithm. The curve, depicted by the blue path, is generated with an initial and goal angles of 90°. . 17

2.9 Illustration of Reeds-Shepp curve, showing the $S^+R^+S^-$ curve which is generated by the *SCS word* of the algorithm. The curve, depicted by the blue path, is generated with an initial angle of 0° and goal angle of 270°. 18

2.10 Illustration of a State lattice for an unstructured environment, showing a grid in which a vehicle can traverse. 19

2.11 Illustration of a State Lattice approach for a structured environment, showing waypoints on a road with one lane. The road may also contain sign boards, which has to be considered while the autonomous vehicle manoeuvres to a goal position. Along the X-axis is the longitudinal road and the Y-axis is lateral road. 19

3.1 Flowchart of actions of the structured road method. The yellow box represents one pulse of input, in terms of given waypoints, to the algorithm. The initial action block, 'Waypoint Processing', approximates the states for every waypoint, addressing the heading and curvature at each waypoint, and increases the number of waypoints laterally along the lane. By receiving the states of the waypoints, the closest longitudinal point from the vehicle, along the road, is extracted. This is followed by path generation from the initial position, to each waypoint neighbour laterally. The red boxes represent frequently received data from the sensors, such as detected obstacles or vehicle states. Trajectories are applied and evaluated whereas the most suitable one is chosen according to a fixed cost function. The wheel angle and desired velocity are output from the algorithm. The green box represents the output. 23

3.2 Illustration of waypoints given along a one-way road, starting from the left-hand side. The solid, black lines represent the road boundaries of a one-way road, the orange arrows represent the heading angle of each point and the grey points represent the given waypoints along a road, *i.e.* the stations. 25

3.3 Illustration of increased number of vertices at each station along the road. The black, solid lines represent the road boundaries, the orange arrows represent the heading angle of each waypoint, the blue points represent the stations, *i.e.* the given waypoints. The grey points represent additional vertices at each station, *i.e.* the augmented waypoints at each station. 26

3.4 Illustration of one generated path, τ_p. The red line represents the path (cubic polynomial) after (p_1, p_2, s_f) have converged to fit the polynomial at station 3 from station 2. The first waypoint on the left-hand side represents station 1. The black, solid lines represent the road boundaries and the grey points represent the given waypoints. 29

3.5 Illustration of generated paths, a set of $\{\tau_p\}$, after (p_1, p_2, s_f) have converged for each vertex at station 3 from the lowest vertex of station 2. The blue points represent the given waypoints, the grey points represent the augmented waypoints. The generated paths do not consider any steering limits. 29

3.6 Illustration of applied velocity profiles as cubic polynomials, to the set of all $\{\tau_p\}$. By applying velocity profiles on the paths, the paths turn into trajectories, seen in the red lines starting from station 2 generated to station 3. A trajectory is denoted as $\tau_t(s)$. Here, the initial velocity is 3 ms^{-1} and the final velocities are equally distributed between 0 ms^{-1} to 10 ms^{-1}. Some trajectories may be unfeasible. The Z-axis represents the velocity, the black lines represent the road boundaries. 31

3.7 Flowchart of Unstructured Environment Method. The yellow boxes represent the inputs at the initial instance. The **Offline Calculation**, mentioned in the top blue box, is an A* algorithm. Path points are extracted from the A* path. Information regarding the vehicle state and the environment, depicted by red boxes, are provided at every instance during the simulation. Following this, the Reeds-Shepp curves are generated to suitable path points based on the current position. The acceleration profile of the vehicle is adjusted by setting a threshold velocity for the vehicle. Replanning of the **Offline Calculation** could be done if the online trajectory generation times-out, as seen in the decision block 'Timeout'. The obtained outputs are wheel angle and acceleration, which are given to the controller of the vehicle, seen as the green box. 36

3.8 Illustration of an unstructured environment, showing obstacles. The green and red markers represent the initial and final positions between which the path is to be generated. The blue and brown boxes represent obstacles in the environment. 37

3.9 Illustration of the A* path, showing the shortest path to the goal by avoiding the obstacle (brown box). Depicted from this figure is that the generated path is holonomic and has sharp turns which are impossible to traverse by a vehicle. 38

3.10 Plot illustrating the information obtained from the A* path, containing the angles of each path point. The purple markers are the path points the are extracted from the A* path. The direction of the nodes, on each path point, is pointed by the red arrows. Each path point carries information about the position and orientation (heading) of the node from the A* path. The green and red markers refer to the initial and final positions. 39

3.11 Plot illustrating the A* path to the sub-point and further extended from the sub-point to the goal. The path points extracted from the total A* path are the purple markers on the A* path. The red dashed line represents the A* path to the sub-point and the blue line is the extended path. The diamond shaped magenta marker indicates the sub-point. Here, the sub-point is placed on the left side of the desired parking spot since the entry direction of the vehicle is seen to be from the right side. 40

3.12 Illustration of a Reeds-Shepp curve generation (the blue line) to the next path point from the initial point. After generating the first curve to the forthcoming path point, the initial position for the next curve is set to be the attained path point. This would result in a continuous path from the vehicle's current position at any instance to the following path point and eventually to the goal. 41

3.13 Plot showing the Reeds-Shepp curve generation (the blue line) to the following look-up path point, since the subsequent path point is closer than a specific distance. The dark green marker represents the position of the vehicle at an instance, assuming it is moving to the 2^{nd} path point but generates the curve to the 3^{rd} path point, since it is near the consecutive path point (*i.e.* 2^{nd} path point). 42

4.1 Illustration of the chosen offline generated trajectories (red lines) from the initial waypoint, station 1 in the left of the plot, to station 12. The best trajectory, to one station forward, is chosen to be used as the initial at next iteration. The black asterisk represents a static obstacle on the road. The trajectory avoids the obstacle and returns back to the centre of the lane. The horizon is 1 station. 47

4.2 Illustration of the chosen, offline generated trajectories (red lines) from the initial waypoint, station 1 in the left of the plot, to station 12. The black asterisk represents an obstacle on the road. The Z-axis represents the velocity. In this figure, the acceleration is smoothed up to the desired velocity, set to $10ms^{-1}$. 47

4.3 Illustration of the chosen, offline generated trajectories from the initial waypoint at station 1 in the left of the plot to station 12, where the best trajectory to 2 stations further is chosen and the end-point is used as the initial point at next iteration. The black asterisk represents a static obstacle on the road. The trajectory clearly avoids the obstacle and returns back to the centre of the lane again. The red lines represent the generated trajectories. 48

4.4 Illustration of the chosen, offline generated trajectories from the initial waypoint at station 1 in the left of the plot to station 12. The black asterisk represents an obstacle on the road. The Z-axis represents the velocity. The desired velocity is set to $10ms^{-1}$. A horizon of 2 stations is used. 49

4.5 Illustration of the chosen, offline generated trajectories from the initial waypoint at station 1 in the bottom of the plot to station 10, upper left, in another scenario. Every 4^{th} station is included, which is another way to extend the distance of trajectories. The black asterisk represents an obstacle on the road. The station horizon is 2. In this case, an obstacle is placed in the middle of the road which the vehicle cannot avoid without adjusting the velocity. 50

4.6 Illustration of the chosen, offline generated trajectories from the initial waypoint at station 1 in the bottom of the plot to station 10. The black asterisk represents an obstacle on the road. The Z-axis represents the velocity. The desired velocity is set to $10ms^{-1}$. A horizon of 2 stations is used. In this case, the vehicle exceeds the threshold limit, $t_{distance}$ and the planned velocity is reduced. 50

4.7 Illustration of the chosen, offline generated trajectories from the initial waypoint at station 1 in the bottom of the plot to station 7. The black asterisk represents an obstacle on the road. A horizon of 2 stations is used. In this case, the vehicle exceeds the threshold limit, $t_{distance}$ and the planned velocity is reduced. The planned trajectory stretches outside the road boundaries due to the long distance between each station. 51

4.8 Illustration of Reeds-Shepp path, generated from an initial angle of 90° to a goal angle of 90°. The green and red triangle markers represent the initial and final positions, while the purple markers are the path points between which the Reeds-Shepp curves are generated. The Reeds-Shepp paths are generated based on the length vehicle which is 3 metres. A safety distance of 10 metres is maintained between the surrounding objects. 52

4.9 Illustration of Reeds-Shepp path along with the A* path. Apart from other parameters, the magenta coloured dotted lines represents the A* path. In this case the safety distance to the obstacles is set to 2 metres. The length of the vehicle is set to 5 metres. 54

4.10 Illustration of Reeds-Shepp path on the A* path indicating the reversing action based on the sub-point generation from the A* path. An observation made from this is that the path points taken from the A* path along with the sub-point are chosen based on the distance mentioned while extracting the path points. 54

4.11 The map simulation environment run in online simulations, illustrating a valet-parking scenario. The ego-vehicle is in the lower end of the image, which is highlighted by a yellow circle. The blue line in front of the vehicle is an example trajectory desired for the whole path, reaching from the initial position to the parking area. The upper right open-space is the entrance to a parking lot. The darker area is the parking lot. The orange line in the centre of the image is the trajectory of a pedestrian crossing the road. 55

4.12 A simplified plot of the road and the parking area, in the simulation environment. The darkest grey square represents the parking lot, the white lines in the parking lot represent the parking spot alignments. The mid-grey square is the entry space from the road to the parking area. The initial position of the road is at $(260, -90)$. The red box is the pedestrian-crossing on the road, the green box is the pedestrian's initial position and the blue boxes are parked vehicles in the parking lot, representing static obstacles. 56

4.13 Illustration of the motion of the vehicle in the simulation environment. The green points represent a moving pedestrian. The pedestrian starts from the top and moves downwards during the simulation. The red box is the pedestrian-crossing on the road, the blue boxes represent the vehicle. The black points are the centre point of the vehicle, at every 40^{th} iteration. The length of the vehicle box is 4 metres. The frequency of the simulation is 20 Hz. The data is extracted for every 2 seconds of the simulation and the vehicle is initialised in the bottom of the plot, moving anti-clockwise. In the blue boxes, the difference in colour-intensity is a function of speed, *i.e.* the higher the speed, the stronger the intensity. The plot demonstrates the vehicle movement after 51^{st} seconds. 57

4.14 Illustration of 59 seconds run-time in simulation. The pedestrian has reached the pedestrian-crossing and the colour-intensity of the vehicle is bright, in front of the crossing. In the last point extraction, the vehicle is reaching stationary state. 58

4.15 A full run-time of the simulation, from start to goal. The colour-intensity of the blue boxes turns very bright in the end, close to the parking area, as the velocity reaches $0ms^{-1}$. The velocity is reduced at the cross-section, at $(225, 20)$. 58

4.16 Illustration of generated trajectories during simulations. The red lines are the trajectories at different iterations, the cyan points represent the trajectories' end. The grey points are the stations, *i.e.* the given waypoints. Points at every 4 seconds are extracted for a total of 76 seconds run-time of the simulation. 59

4.17 Illustration of generated trajectories during simulation of 76 seconds. Now, points are extracted from every second of the simulation. 60

4.18 Illustration of the generated trajectories, from start to end. Points are extracted every 4^{th} second. The black arrows are indicating every 50 metres of the road. 60

4.19 Illustration of the velocity behaviour for the whole track. The velocities are extracted every second. Comparing this plot with Figure 4.18, it is seen that the velocities are reduced in the curves, although, not early enough due to the short horizon. 61

4.20 Illustration of the generated trajectories including velocities. The trajectories are extracted from initial position to the first half of the route, *i.e.* from $(260, -90)$ to $(210, 30)$ in Figure 4.18. The samples are extracted from every second. The black points represent the location of the vehicle, the blue points represent the end of each trajectory and the red lines are the trajectories. The initial velocity is $0ms^{-1}$ and reaches up to $5ms^{-1}$ at maximum. 62

4.21 Illustration of the generated trajectories including velocities. The trajectories are extracted from 2^{nd} half to end, *i.e.* from $(285, 14)$ to $(400, 120)$ in Figure 4.18. The samples are extracted from every second. The vehicle brakes before the moving obstacle. 62

4.22 Illustration of the trajectories generated in the simulation environment. The trajectories generated in the simulation are at every 4^{th} second, from each path point. The green and red markers represent the initial and final point of the vehicle. The initial position is $(400, 122)$ with an angle of $0°$ and the final position is $(529, 125)$ with a desired angle of $270°$. The blue boxes are static obstacles present in the parking environment. 64

4.23 Illustration of the path taken by the vehicle to the goal. It is seen that the path is followed based on the trajectories generated from the algorithm. By setting an saturation limit to the wheel angle, the vehicle can be avoided from taking sharp turns. In this depiction, it successfully avoids the crossing of the priority lines and safely enters into the desired position. The offline path obtained is first generated to the sub-point and then to the goal. In this way, any object present around the sub-point is also avoided. The purple circled markers indicate the path points obtained offline. 65

4.24 Illustration of the vehicle position and its path for the next 40 metres from that position until it reaches the goal. The magenta markers denote the position on the chosen path at every 4^{th} second. 66

4.25 Illustration of the velocity of the vehicle along the path. It is observed that the negative velocity refers to the reverse motion of the vehicle in the simulation environment. The overall depiction of the velocity is with respect to the arc length along the path. The threshold on velocity set for this case is $\pm 3ms^{-1}$. 66

A.1 Illustration of the generated trajectories in a difficult case, the corner. Point extraction frequency of 1 second. I

A.2 Illustration of the generated trajectories from start to end. Points are extracted every 1 second. II

A.3 Illustration of paths generated at a time interval of 2 seconds in the PreScan simulation environment. III

A.4 Illustration of the path obtained in the simulation environment. The initial point of the vehicle is at (262,-93) and the final point is at (529,125) on the plot. Both the algorithms are merged and a smooth transition is seen during the switch of the algorithms. The waypoints for the structured road are highlighted by the white dots and the path points for the unstructured environment are highlighted by magenta circles. The change of algorithm takes place at (398,122) on the plot. IV

B.1 Illustration of RRT algorithm, showing an example of random node search mentioned as x_{rand} and many near nodes (x_{near}) expanding bidirectionally with a stepsize between x_{start} and x_{goal}, seen as red points. Here, the exploration is seen to be in a three dimensional environment. In this figure, the obstacle is the orange, oval shape in the middle of the map. Image taken from [33]. VI

B.2 Illustration of Dubins' Path showing a forward manoeuvre, from the start node to the goal node, given the initial angle (0°) and goal angle (180°). Here, the blue line depicts the path, the initial and final positions are represented by the red and the green circle, respectively. VII

List of Tables

3.1 Cost function parameters for generated paths 32
3.2 Cost function parameters for the dynamic cost 33

4.1 Weights used in Cost Function. 46
4.2 Parameters for Test Scenario 1. 52
4.3 Parameters for Test Sccenario 2. 53
4.4 Parameters for Test Scenario 3. 55
4.5 Parameters for Simulation Test. 63

1
Introduction

The idea of having autonomous vehicles emerged in the year 1925, invented by Francis Houdina; yet it was not until the year 1969 that the idea of autonomous vehicles rose again due to John McCarthy - one of the fathers of Artificial Intelligence. Since then, a lot of focus has been oriented towards autonomous vehicles, specifically in the field of Robotics, Computer Science and Engineering. With the increase of processing power and artificial intelligence, the implementation has been realised in on-road vehicles. Three major milestones in the development of autonomous vehicles are the DARPA Challenges in years 2004, 2005 and 2007. The first challenge summarised a $230km$ long, off-road track to be completed autonomously as fast as possible. Although, no competitor was able to complete the first race, five of twenty three teams finished within ten hours in the $212km$ long desert terrain track of the DARPA Grand Challenge 2005. In the DARPA Urban Challenge (2007), six teams completed the race in an urban environment indicating the possibilities of urban autonomous vehicles, contributing to the state-of-the-art development of today's algorithms.

In the year 2016, there were almost 1.1 million reported accidents in European Union, whereof 25.6 thousand led to fatal consequences [10]. According to the National Highway Traffic Safety Administration (NHTSA), 37461 people died in a traffic in USA, of which 28% were alcohol-impaired, 9.2% were distraction affected crashes and 2.1% involved drowsy drivers [11]. An estimated rating of 94% of collisions in USA are associated to drivers' error [12]. Numerous benefits can be introduced by including autonomous features in a vehicle on the roads, among the major factors, such as safety, time saving.*e.g.* searching for free parking spots [13], stress reduction and fuel efficiency.

1.1 Background

Autonomous vehicle systems are generally composed of three essential stages. An example of the hierarchy can be seen in Figure 1.1. From a high-level point of view, in the first stage the vehicle system collects and pre-processes information through numerous sensors (*e.g.* LiDAR, RADAR, Cameras, GPS). The processed information is fused to obtain a good perception of the environment, estimation of states, localisation and classification of objects and object-motion prediction. The system creates a perception and environment model of the world.

Using the perception and environment model retrieved from the first stage, the second stage comprises of decision making along with the provision of planned paths

and generated trajectories for the vehicle. The generated trajectories shall respect the constraints of the vehicle's kino-dynamics (kinematics and dynamics), be feasible (achievable), satisfy the passengers' comfort and perform manoeuvres that are safe for the passengers, as well as for the environment. Concurrently, the rules and laws shall be obeyed as the vehicle navigates to the desired destination.

Typically, the last stage of an autonomous system includes the local motion controller of the vehicle. After obtaining data from the first and second stages, the vehicle shall act according to the generated trajectories, where the controller is responsible for actuating the brakes, throttle and steering wheel to follow the trajectories.

All the three stages are crucial and are dependent on each other. In this thesis, the focus is narrowed down to the second stage, specifically in path planning and trajectory generation; a memory consuming and computationally heavy process. The foundation of this planning is derived from mobile robotics, yet one major issue is the non-holonomic nature of a vehicle, defined in Section 4.

Path planning and trajectory generation is approached with various techniques. In some techniques, the path planning and trajectory generation is computed as a single entity, while in other techniques the trajectory generation is a process that is posterior to path planning. Stage two can be divided into three or four classes. *For example*, the top-level represents Route planning, a global route from current position to destination acting as a reference for the lower level classes. Subsequently, the other levels represent Decision making and Path/Trajectory planning, *i.e.* deciding the manoeuvre of the vehicle, such as overtaking another vehicle, stopping or turning. The scope of this thesis is focused towards path planning and trajectory generation. Route planning and decision making are excluded.

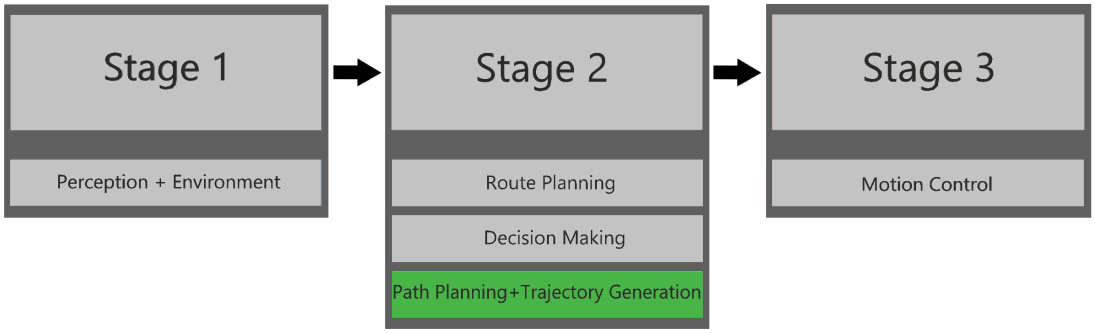

Figure 1.1: Example of the architecture of an autonomous system.

1.2 Aim

The purpose of the project is to demonstrate a valet parking scenario where, in theory, a driver exits the vehicle and the vehicle proceeds autonomously towards a parking lot and parks itself in a free parking space. The project is divided into two sub-parts.

The first part implicates to accomplish an autonomous navigation algorithm, for a vehicle in a structured road simulation environment. A structured road implies a road with lanes including alignments. The intention is to frequently generate trajectories for the vehicle to act upon accordingly, that is geometric reference paths along with velocity applications. This is done from the initial position at a given location to a desired parking area. The performed action derived from the trajectories should result in a way such that the driverless vehicle avoids static and dynamic obstacles, execute feasible actions, stays within the road boundaries and travels below or equal to the speed limit. The autonomous vehicle shall operate safe as well as comfortable for passengers and the surrounding environment.

The second part consists of developing an algorithm that parks the vehicle autonomously, by generating frequent paths and trajectories that are feasible, comfortable and safe. The algorithm should be developed such that the system is able to park the vehicle both forward or reverse. The desired parking spot (including position and orientation) will be given. By regulating the vehicle according to the given trajectories, the parking action should result in a collision free motion in an environment including obstacles. In the second part of the thesis, the environment is assumed to be unstructured, thus there are neither lanes nor alignments in the environment. The system should find the shortest way to the given parking spot, without crossing other parking spots.

1.3 Scope and Limitations

The scope of the thesis is fulfilled based on the software tools; Matlab, Simulink and PreScan. The development of the algorithms are limited to Matlab where evaluation of the code is done offline. The real-time testing and evaluation of the algorithms is done in PreScan with Simulink as a foundation tool. The scope includes performances in generic scenarios. The scenarios are created by the authors of the thesis.

The thesis is primarily limited to path planning and trajectory generation algorithms. The default controller of the vehicle (steering angle, acceleration and braking) is modified in the simulation software, in order to try the different algorithms. This is not the main focus of the thesis.
As the aim is targeting stage two of the three stages, stage one which includes the sensor framework, is considered to be ideal and adhering the ground truth. Accurate coordinates and velocities of the ego-vehicle, exact obstacle locations, the map data and lane boundaries are accessed through the simulation environment. The structured roads are assumed to be symmetric and plane.

On the structured roads, waypoints including (x, y) coordinates are given to the vehicles manually, approximately every 2^{nd} metre from initial position to the parking lot. The waypoints are placed in the centre of the desired lane to follow. In the parking lot, the desired parking spot is acquired as one final destination point with the initial and final vehicle coordinates (x, y) and vehicle orientation (θ). The goal in the unstructured environment is to generate the points of the shortest path from the initial position of the parking area to the desired parking spot and autonomously

generate trajectories between the points at every instance until the vehicle reaches the destination. As mentioned, both these algorithms should generate trajectories that avoid obstacles.

Once the vehicle (a car, a truck or a bus) has entered the parking lot, a switch between the algorithms shall be performed automatically. This part comes under decision-making and so, it shall not be deeply included to evaluate the results of this project. Further, the limitations are rules and laws of the environment. Speed-limits and road boundaries shall be considered, whereas other rules as right-lane rule or vehicle precedence shall not be taken into account as this is generally a part of decision-making layer in an autonomous system. Therefore, overtaking of other vehicles will be disregarded.

Another limitation to be encompassed is the dimension of the environment. The vehicle is limited to move in two dimensions, (x, y). In simulation, the dynamics of the vehicle is modelled as a bicycle model. The dynamics are developed by the developer of PreScan and handles the kino-dynamic actions of the vehicle based on various inputs such as wheel angle, acceleration and braking.
The methods should work for both cars and trucks. The difference between these vehicles are the dynamics.

1.4 Thesis Outline

The outline of the remaining chapters of this thesis is as follows: Chapter 2 provides theory regarding the definitions and theoretical background about different concepts, as a preparation for the remainder of the thesis. The search techniques and planning algorithms are briefly presented. In Chapter 3, the techniques adapted for this thesis are presented in detail, with the first part describing the navigation algorithm on a structured road, *i.e.* driving to the parking lot, and the second part describing the parking algorithm of the vehicle. Chapter 4 presents the results and parameter values obtained by implementing the algorithms in the simulation environment. The behaviour of the vehicle in different scenarios is illustrated. Further, Chapter 5 reflects the observations made from the results. The pros and cons of the chosen algorithms are emphasised. Finally, in Chapter 6, the overall thesis is concluded with a reflection of the results to the aim. A brief motivation for future work is also mentioned.

2
Theory and Related work

In this chapter, definitions and mathematical preliminaries that are used in the context of this thesis are introduced in Section 2.1 and Section 2.2. The vehicle model used for this thesis is expressed in Section 2.3. The remainder of this chapter consists of various search techniques and algorithms used for path planning and trajectory generation problems, whereas the theory behind some of the base algorithms is presented in depth. The last two sections of this chapter, Section 2.5 and Section 2.6, explain about the basic principle followed in this thesis. The important algorithms followed are State lattice including cubic polynomials, A* path planning and Reeds-Shepp's curve generation.

To complete a path planning and trajectory generating method, different algorithms are fused together. There are numerous methods of performing these actions. Different authors use different combinations of methods depending on the scenario, *for instance*, structured, unstructured or semi-structured environments.

2.1 Theoretical Preliminaries and Definitions

The differentiation between path planning and trajectory generation is mainly done by the time parameter. Path planning is based on finding a feasible path in a geometric and spatial sense. Trajectory generation includes time and velocity parameters. These are used to decide a suitable speed throughout the manoeuvre considering the dynamics of the vehicle based on the geometric path.
Paths and trajectories are defined differently by various authors. In the context of this thesis, path and trajectory are defined as in [2] and [1], seen in Definition 1, Definition 2 and Definition 3.

Definition 1 *The **configuration space** (C-space)*

$$C = \{(x, y, \theta, \kappa)\}, \tag{2.1}$$

implies that all the possible states which a robot can have, can be comprised as a set C. (x, y, θ, κ) represents the robot coordinates, heading and curvature respectively.

Definition 2 *A **path** is defined as a continuous sequence of configurations starting with an initial configuration and ending with a goal configuration.*

Definition 3 *A **trajectory** is a path with explicit parameterisation of time, that is velocity.*

Definition 4 *A **holonomic** system is the one in which the number of controllable degrees of freedom is equal to the total degrees of freedom, for a given configuration space.*

Definition 5 *A **geodesic** is relating to or denoting the shortest possible line or arc between two points on a sphere or a curved surface.*

2.1.1 Path Planning

With reference to Definition 4, a moving car-like robot is non-holonomic as it cannot move laterally without also moving longitudinally. The heading has to be considered in order to point out the possible movement that the robot can have from a certain state.

With reference to Definition 1 and Definition 2, path planning is the problem of finding a geometric path in set C from an initial configuration to a final configuration while also satisfying the global and local constraints. These constraints are, *for example*, the non-holonomic nature along with obstacles. The complications of path planning is to generate a feasible path while respecting the constraints. Additional constraints may be to acquire smoothness of the path.

2.1.2 Trajectory Generation

Trajectory generation refers to the vehicle transition from a feasible initial configuration to the next configuration including time as a parameter. A trajectory is produced from a path by applying a time-parameterised quantity such as acceleration or velocity. Once a trajectory has been generated, it is possible to evaluate its characteristics in terms of smoothness and safety.

2.2 Mathematical Preliminaries

The mathematical background of the algorithms presented in Chapter 3, are showcased in this section. Further calculations that are explicitly applied to the trajectory generation are presented in the above mentioned chapter.

2.2.1 Composite Simpson's Rule

The Composite Simpson's Rule is used to approximate an integral numerically. The Rule is written as

$$\int_a^b f(x)dx \approx \frac{h}{3}\Big[f(x_0) + 2\sum_{j=0}^{\frac{n}{2}-1} f(x_{2j}) + 4\sum_{j=1}^{\frac{n}{2}} f(x_{2j-1}) + f(x_n)\Big], \tag{2.2}$$

where $x_j = a + jh$ for $j = 0, 1, ..., n$. $h = \frac{b-a}{n}$, $x_0 = a$, $x_n = b$. n is the number of evenly spaced sub-divisions in $[a, b]$.

2.2.2 Newton-Raphson Method

Newton-Raphson's Method is a method for numerically finding the roots of a function, iteratively. For non-linear systems, Newton-Raphson method is presented as

$$x^{n+1} = x^n - J_F(x^n)^{-1}F(x^n)m, \tag{2.3}$$

where, x^n is the current approximation and x^{n+1} is the next approximation. Provided the current approximation, *i.e.* the initial guess, the method usually converges to 0. J_F represents the Jacobian of the function. m is a scalar representing the step-size of the convergence, ranging between 0 and 1.

2.3 Vehicle Model

To approximate the behaviour of a car-like robot, a proper vehicle model has to be chosen. While a high-fidelity model may reflect a vehicle's response rigorously, the complexity of such a model yields a more complicated planning for the path planner. A solution has to be found such that the planner is simple, but also the vehicle model matches the behaviour of the vehicle's response. In terms of the kinematics of a bicycle model, there is a fixed rear wheel along with an extra degree of freedom on the front wheel, hence it reflects a car in an approximated manner. By also assuming no-slip on the vehicle, the model gets simplified yet stays accurate enough to describe the motion and the non-holonomic nature.
From Figure 2.1, introducing no-slip assumption, similar to [2], g_r and g_f are assumed to be collinear with the corresponding wheel orientation. To calculate the motion of the rear-wheel, which is the reference point in this thesis, the coordinates are expressed as

$$x_r = g_r \cdot \hat{e}_x, \tag{2.4a}$$

$$y_r = g_r \cdot \hat{e}_y, \tag{2.4b}$$

where x_r and y_r denote the current coordinates of the vehicle, in the global Cartesian frame. g_r and g_f are the contact points on the ground, for the rear and front wheels. $\hat{e}_x$ and $\hat{e}_y$ are the unit vectors along the X and Y axes. Calculating the vehicle speed, v_r, which is the magnitude of $\dot{g}_r$, yields

$$v_r = \dot{g}_r \frac{g_f - g_r}{||g_f - g_r||}. \tag{2.5}$$

By having the rear wheel as a reference point, the kinematics in terms of (x, y, θ) is presented as

$$\dot{x}_r(t) = v_r(t)cos(\theta_r)(t), \tag{2.6a}$$

$$\dot{y}_r(t) = v_r(t)sin(\theta_r)(t), \tag{2.6b}$$

$$\dot{\theta}_r(t) = v_r(t)\kappa_r(t), \tag{2.6c}$$

where θ_r represents the heading angle of the ego-vehicle. κ_r is the curvature, which is the rate of change as a function of distance, for the ego-vehicle. The Equation (2.6a) to Equation (2.6c) are calculated with respect to time, t. The planning approach in

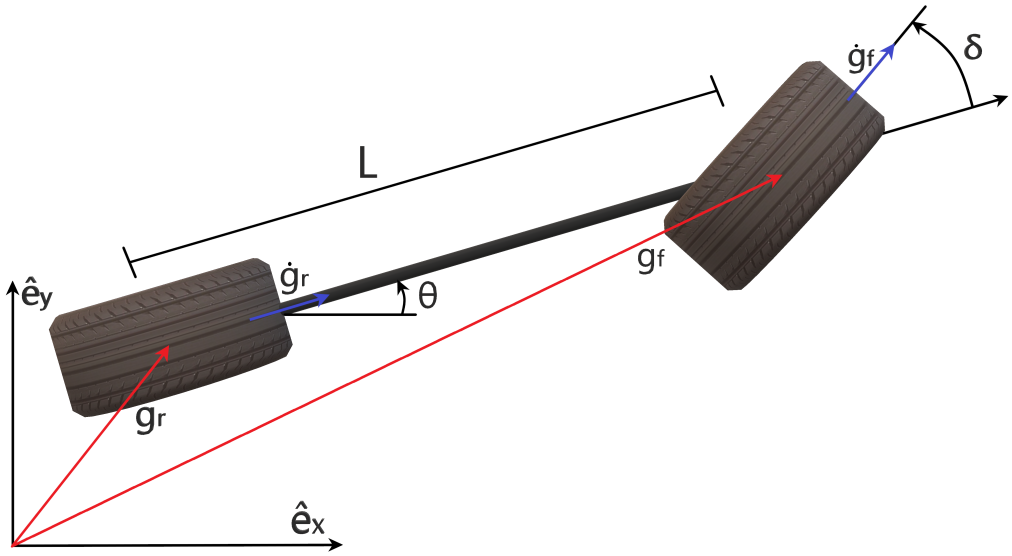

Figure 2.1: Kinematic bicycle model with no-slip assumption, as seen in [2]. L is the length between the front and rear axle. δ is the front wheel angle, θ represents the heading angle of the vehicle, *i.e.* $g_f - g_r$. g_r and g_f indicate the ground contact point of the rear and front wheel, respectively. $\hat{e}_x$ and $\hat{e}_y$ are unit vectors in (x, y) coordinates.

this thesis separates the path planning and the trajectory generation. This implies that, firstly, the geometrical paths are generated and then the speed trajectories are applied.
By deriving the motion equations with respect to the arc length, s, the equations with respect to arc length are expressed as

$$\frac{dx_r}{ds} = cos[\theta_r(s)], \tag{2.7a}$$

$$\frac{dy_r}{ds} = sin[\theta_r(s)], \tag{2.7b}$$

$$\frac{d\theta_r}{ds} = \kappa_r(s). \tag{2.7c}$$

The path model is presented as

$$x_P(s) = \int_0^s cos[\theta_P(s)]ds, \tag{2.8a}$$

$$y_P(s) = \int_0^s sin[\theta_P(s)]ds, \tag{2.8b}$$

$$\theta_P(s) = \int_0^s \kappa_P(s)ds, \tag{2.8c}$$

$$\kappa_P(s) = \frac{\tan(\delta(s))}{L}. \tag{2.8d}$$

$(x_P, y_P, \theta_P, \kappa_P)$ are the states of a path. δ is the wheel angle and L is the length between the rear and front wheel axles. The vector of these states is collectively expressed as

$$X_P(s) = [x_P(s)\ y_P(s)\ \theta_P(s)]. \tag{2.9}$$

Seen in Equation (2.8d), the curvature and wheel angle are dependent on each other. By setting the maximum and minimum wheel angle, the curvature is constrained as

$$\kappa_r \in \left[\frac{tan(\delta_{min})}{L}, \frac{tan(\delta_{max})}{L}\right]. \tag{2.10}$$

2.4 Planning Algorithms

In this thesis, the path planning and trajectory generation algorithms are approached in two ways. The first goal, among the two presented in Section 1.2, is considered in structured environments, where the alignments are fixed to the road. The main idea is to be able to navigate on a road from the initial position to a parking lot. The second goal is considered in unstructured environments, where the vehicle should navigate to the desired spot in an open space, such as a parking depot.

As this is an important part of an autonomous system, a lot of research in this field is carried out, leading to a huge diversity of different approaches. From the results of the DARPA Urban Challenge (2007), the top four autonomous cars had different solutions to solve these problems. Some approaches were split up by first finding a feasible or optimal path, followed by the trajectory generation. Other approaches included the trajectory generation as a part of the path planning search. In these approaches, the paths were generated using different search techniques in the configuration space.

One of the first search techniques presented in the context of shortest path algorithms is Dijkstra's algorithm. Another more efficient technique for finding the shortest path to a fixed point from an initial position is the A* (pronounced as A-star) algorithm. This is computationally more efficient than Dijkstra's algorithm. The first step of executing this algorithm is by discretising the configuration space, followed by which the vehicle path is generated based on different criteria in discrete steps.

Some common path search algorithms are Dijkstra's, A* and Rapidly-exploring Random Trees (RRT). Advancements of these algorithms, such as Dynamic A* (D*), Anytime A*, Weighted A*, RRT* and many other variants are focused towards specific applications. A common way of using a path search algorithms is to first generate the shortest path to a goal which avoids obstacles, and then post-process the path, such that it becomes feasible with respect to the kinematics of the vehicle. A brief classification of path-finding algorithms can be found in Appendix B.1.

2.4.1 Working of A* Algorithm

In the Dijkstra's algorithm, from [17], the path from an initial node to the goal node is obtained by collecting the nodes to the goal in a queue. The queue has nodes placed in an order of priority. In general, the priority is based on the weight or cost assigned to every node. Further improvements in the above mentioned algorithm led to the A* algorithm, as seen in [18], which is a basis of the second part of the thesis. The A* algorithm has a working principle similar to Dijkstra's algorithm. The improvement in this method is that the cost to every neighbouring node is evaluated based on a heuristic from every neighbour to the goal. Thus, this method has proven to be an efficient path planning candidate in terms of avoiding obstacles and still achieving the shortest path.

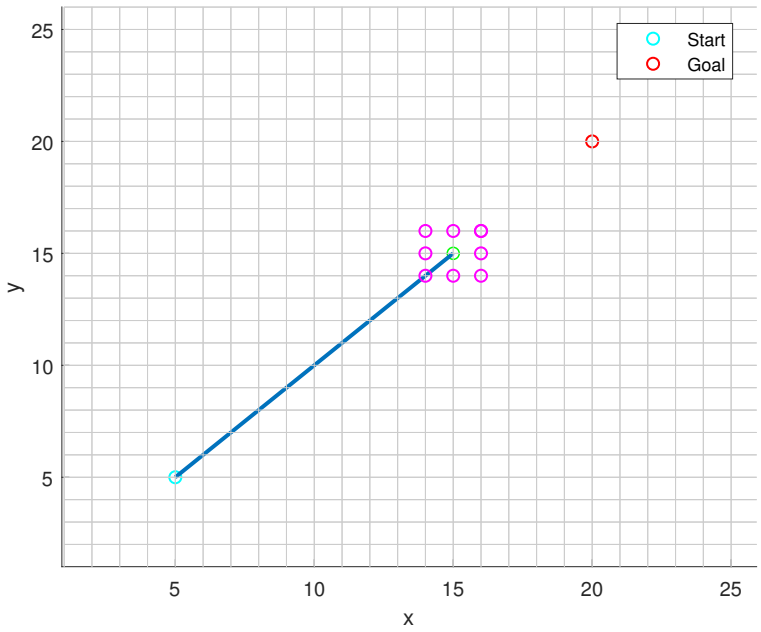

Figure 2.2: Illustration of A* path, showing all possible neighbouring nodes to the current node. Here, the cyan circle at $(5, 5)$ is the initial node. The green circle at $(15, 15)$ is the current node and the red circle is the goal node, the magenta circles represent the neighbours surrounding the current node. The X and Y axes represent the dimension of the map.

The working of the traditional A* algorithm is explained as following. The goal of this algorithm is to search for the shortest path from the initial point to the final point. The neighbour-based search approach involves checking of the cost from current node to each neighbour node and from each neighbour to the final node. There are two major lists in the A* algorithm that keep track of the visited nodes. They are *Open List* and *Closed List*. Initially, the *Open List* consists of the initial node, while the *Closed List* is empty. From the start node, until the final node is reached, each node is checked based on a cost calculated by the cost function. The cost function includes weighting of parameters such as the distance to the goal or a safety distance from an obstacle for each neighbour node on the provided map. The

two costs that contribute to the cost function are the actual cost, which penalises the node to move from current node to the corresponding neighbour, and the heuristic cost, which penalises a node based on the distance to move from the neighbour to the goal node. The heuristic cost is usually the *Euclidean distance*, *Manhattan distance* or a similar mathematical weighing function. A neighbour-based search is carried out, in which usually eight neighbours surrounding the current node are compared, based on the cost from each neighbour node to the goal node, as seen in Figure 2.2. The decision to choose a neighbour node is by prioritising the nodes based on the least cost, shown in Figure 2.3. Hence, the search continues and the information regarding the position, orientation and total cost of each node is stored in the different lists. All the visited nodes are placed in the *Open List*, while the nodes with the optimal solution leading to the goal are moved to the *Closed List*. The path obtained by this search method can be seen in Figure 2.4 and Figure 2.5.

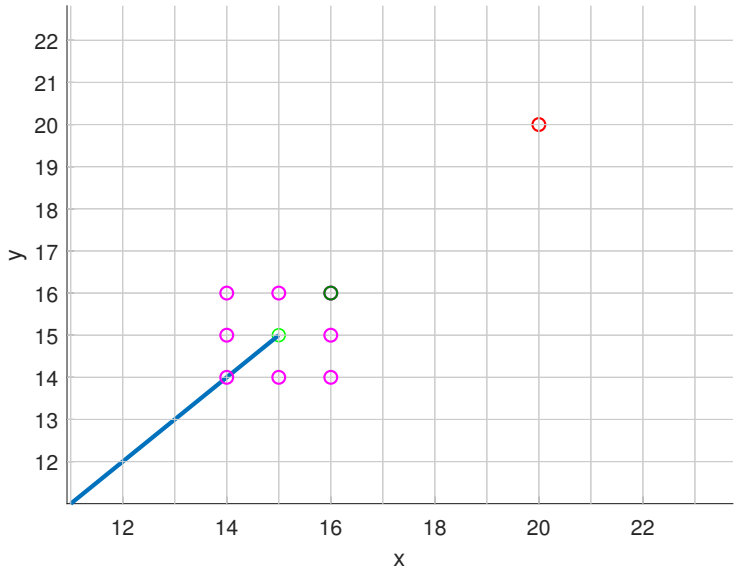

Figure 2.3: Zoomed view of the A* path calculating the next step. This figure shows the chosen node among the neighbour nodes, from the current node. The green circle is the current node and the red circle is the goal node, the dark green circle is the chosen node added to the path, towards the goal node. The X and Y axes represent the dimension of the map.

2.4.2 Explanation of Reeds-Shepp Algorithm

The above explained algorithms provide paths from one point to another without considering the kinematics of a vehicle. This leads to generation of non-traversable paths, since a vehicle cannot turn without moving longitudinally. To overcome this constraint, different methods have been developed. Two of the foremost methods of generating geometrical paths between two points are the Dubins' path and Reeds-Shepp's path, where the latter method is used as basis in the second part of this thesis. These approaches include the fact that a vehicle is non-holonomic and there-

fore, generate paths by forming arcs to reach a goal position. In this subsection, the Reeds-Shepp approach is explained. The explanation of Dubins' method can be seen in Appendix B.2.

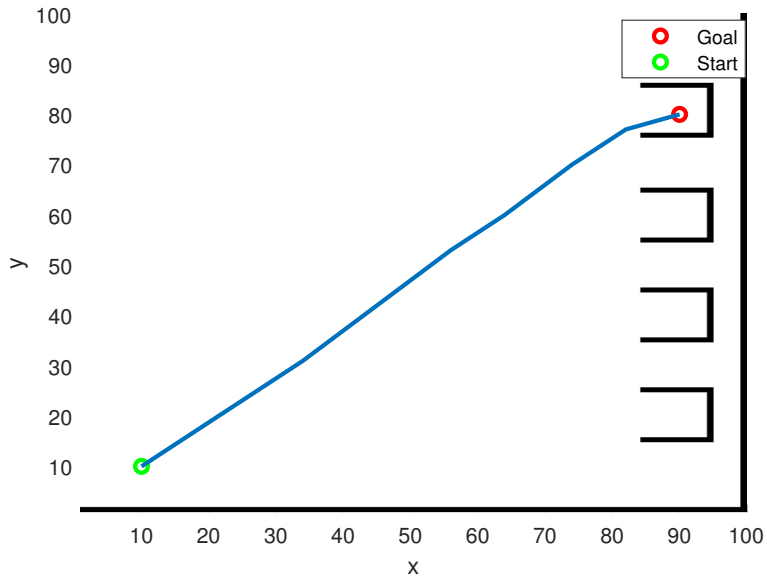

Figure 2.4: The blue line illustrates a path by A* algorithm, showing the shortest path between two points. The goal is depicted to be inside the parking place, which is represented by the red circle. An A* path is generated from the initial position, represented by the green circle, to the goal.

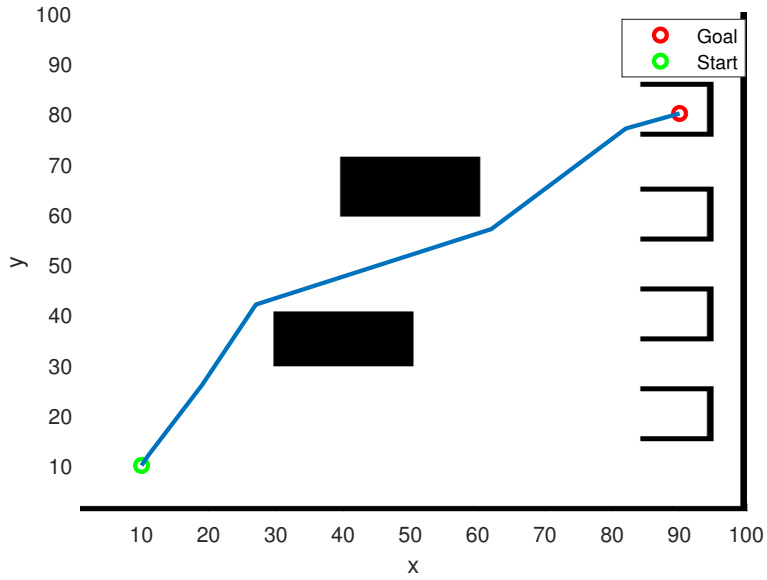

Figure 2.5: Illustration of the A* path, showing that the path also avoids obstacles, placed in-between the environment (the black boxes).

2.4.2.1 Reeds-Shepp's method

In order to implement reverse parking in a vehicle, the vehicle has to be able to plan a path that is the shortest and will allow the vehicle to manoeuvre forwards and backwards. Reeds-Shepp's model is implemented to obtain a path for the vehicle to move forwards and backwards. Considering the kinematics of a vehicle model (bicycle model), which is subjected to non-holonomic constraints, this method should result in providing the shortest and feasible path. This method neither requires holonomic path planning nor does it require construction of the configuration space. The only necessary states are the positions (initial and final coordinates) and the heading angles (initial and final orientation). The working of this method is similar to that of Dubins' method, but the scalar equations used to calculate the parameters, that represent the length of the curves, are slightly different, as showcased in [31].

Explained in [32], a brief working of this method is as follows:

- The initial and final coordinates are obtained in the Cartesian coordinate system.
- Using the scalar equations, Equation (2.20) to Equation (2.27), the parameters corresponding to the length of the segments are obtained. During this, the segments of the curve are decided to be forward or reverse depending on the required orientation of the vehicle at that point.
- This is carried out for each curve until the goal point is reached with the desired orientation.
- Finally, the curves are interpolated to obtain a smooth and traversable path which also consists of the reverse curves.

The preliminary *Lemmas* to generate the geometric curves can be found in Dubins' primary study of generation of curves. These obey the non-holonomic constraints of a car-like robot, [27]. The Reeds-Shepp equations defined for a vehicle to move forwards and backwards are found in Reeds-Shepp's study for a vehicle to move forwards and backwards.

A path of a vehicle between two points is bound by its radius of curvature, cusps, and change of direction (forward or reverse). Therefore, to move between the points and also have the shortest possible distance, a path can be defined by calculating the length of all possible paths with all possible cusps that the vehicle can make to have a proper manoeuvre. The generated path from the vehicle's current position and orientation, (x_r, y_r, θ_r), at any instance s, where s is the arc length of the path, can be obtained from

$$x_P(s) = x_r + \int_0^s \epsilon(s) cos\theta_P(s) ds, \tag{2.11}$$

$$y_P(s) = y_r + \int_0^s \epsilon(s) sin\theta_P(s) ds, \tag{2.12}$$

$$\theta_P(s) = \theta_r + \int_0^s \kappa_P(s) ds, \tag{2.13}$$

where, (x_P, y_P, θ_P) are the coordinates and orientation along the generated path. κ_P is the curvature. These equations are formulated using the Equation (2.8a) to

Equation (2.8d). ϵ represents the forward or reverse motion of the vehicle on the path.

A path is divided into curves. Each curve consists of at least three segments. A combination of segments gives the geodesic to reach a goal point. Each segment may consist of an arc of a unit circle of minimum turning radius, represented as *C*, or a line segment, represented as *S*. The combination of segments or cusps is referred as ***'word'***. *For example*, a *word* may be *CCC*, which represents a curve with three segments consisting of three cusps. A *word*, therefore, is a finite string of segments. In Reeds-Shepp's paths, cusps are considered such that each geodesic has '+' or '-', representing forward or reverse motion on the path. It is possible to have a curve with more than three cusps. Each segment has a parameter that gives an angle of the arc on a unit circle or length of the line segment from the initial point or the point of cusp. The position and orientation are calculated using the following notations,

$$L_\lambda : \mathcal{R}^3 \rightarrow \mathcal{R}^3, \tag{2.14}$$

$$R_\lambda : \mathcal{R}^3 \rightarrow \mathcal{R}^3, \tag{2.15}$$

$$S_\lambda : \mathcal{R}^3 \rightarrow \mathcal{R}^3, \tag{2.16}$$

in which λ is the parameter that is used to represent either the length of the current segment in case of a straight line, (S) or the final angle of the arc (L, R) on a unit circle.

According to [31], a curve (CSC, CCC or similar) starting from a point (x_r, y_r, θ_r), must end at (x_g, y_g, θ_g). The segments to reach this position are calculated based on general scalar equations such that,

$$L_\lambda(x_g, y_g, \theta_g) = \Big(x_r + sin(\theta_r + \lambda) - sin(\theta_r), y_r - cos(\theta_r + \lambda) + cos(\theta_r), \theta_r + \lambda\Big), \tag{2.17}$$

$$R_\lambda(x_g, y_g, \theta_g) = \Big(x_r - sin(\theta_r - \lambda) + sin(\theta_r), y_r + cos(\theta_r - \lambda) - cos(\theta_r), \theta_r - \lambda\Big), \tag{2.18}$$

$$S_\lambda(x_g, y_g, \theta_g) = \Big(x_r + \lambda cos(\theta_r), y_r + \lambda sin(\theta_r), \theta_r\Big), \tag{2.19}$$

where, (x_r, y_r, θ_r) represents the initial point of the curve and (x_g, y_g, θ_g) represents the final point of the curve.
Similar to Equation (8.1) to Equation (8.3) of [31], the equations to calculate the parameters based on the type of *word* are given by

$$\lambda_1 = tan^{-1}\Big(\frac{y_r - 1 + cosd\theta}{x_r - sind\theta}\Big), \tag{2.20}$$

$$\lambda_2 = \sqrt{(x_r - sind\theta)^2 + (y_r - 1 + cosd\theta)^2}, \tag{2.21}$$

$$\lambda_3 = d\theta - \lambda_1, \tag{2.22}$$

for a curve of type **CSC**, the parameters of *lsl* are shown to be calculated, and for a curve of type **CCC**, the parameters of *lrl* are calculated as,

$$\bar{\lambda_1} = tan^{-1}\Big(\frac{y_r - 1 + cosd\theta}{x_r - sind\theta}\Big), \tag{2.23}$$

$$\bar{\lambda_2} = \sqrt{(x_r - sind\theta)^2 + (y_r - 1 + cosd\theta)^2}, \tag{2.24}$$

$$\lambda_2 = -2sin^{-1}\Big(\frac{\bar{\lambda_2}}{4}\Big), \tag{2.25}$$

$$\lambda_1 = \bar{\lambda_1} + \frac{\lambda_2}{2} + \pi, \tag{2.26}$$

$$\lambda_3 = d\theta - \lambda_1 + \lambda_2. \tag{2.27}$$

It is understood that, *for example*, a curve, say $L_{\lambda 1}S_{\lambda 2}L_{\lambda 3}$, starting at $(0, 0, 0)$, where $\lambda_1, \lambda_2, \lambda_3$ are the parameters representing the lengths of segments or angles of the arcs of a unit circle, must end at (x_g, y_g, θ_g). The segments for this curve are calculated as $L_{\lambda_3}(S_{\lambda_2}(L_{\lambda_1}(0, 0, 0)))$, by substituting the parameters in Equation (2.17) to Equation (2.19). This can be obtained such that, initially,

$$L_{\lambda 1}(0, 0, 0) = (sin\lambda_1, -cos\lambda_1 + 1, \lambda_1), \tag{2.28}$$

In physical terms, the point attained by performing a turn (Left), for a parameter λ_1, is given above. Following this, the next segment is calculated as

$$S_{\lambda 2}(L_{\lambda 1}(0, 0, 0)) = (sin\lambda_1 + \lambda_2 cos\lambda_1, -cos\lambda_1 + 1 + \lambda_2 sin\lambda_1, \lambda_1). \tag{2.29}$$

Finally,

$$\begin{aligned} L_{\lambda 3}(S_{\lambda 2}(L_{\lambda 1}(0, 0, 0))) = (&sin\lambda_1 + \lambda_2 cos\lambda_1 + sin(\lambda_3 + \lambda_1 - sin\lambda_1), \\ &-cos\lambda_1 + 1 + \lambda_2 sin\lambda_1 - cos(\lambda_3 + \lambda_1) + cos\lambda_1, \lambda_1 + \lambda_3). \end{aligned} \tag{2.30}$$

The observation made from this is that, the change in angle of this curve is contributed by the parameters, λ_1 and λ_3, as mentioned in Equation (2.22). The relation between arc length and angle of a unit circle is expressed as

$$\theta_r = \frac{l}{r}, \tag{2.31}$$

where θ_r is the heading angle of the vehicle, l is the length of the arc and r is the radius of the circle. Thus, $\theta_r = l$, for a unit circle. Therefore, the parameters λ_1 and λ_3 indicate angles of the arcs (Left) and λ_2 indicates the length of the line segment (Straight).

As it is seen that, in some cases, the end conditions are less than the number of parameters to be calculated. This refers that one of the parameters is redundant, which means that it is either equal to one of the other three parameters or has an arbitrary value. Reeds-Shepp's article presents 48 formulae that suffice to generate path for any given case, as shown in Table 1 of [31]. According to the *Lemmas* stated by Reeds-Shepp,

- A *word* of the form *SCS*, *SCC*, *CCCC* can be shortened to a word of the form *CSC* or *CCC*,
- A *word* with more than two segments having less than two cusps must have one of the parameters equal to $\pm\frac{\pi}{2}$.

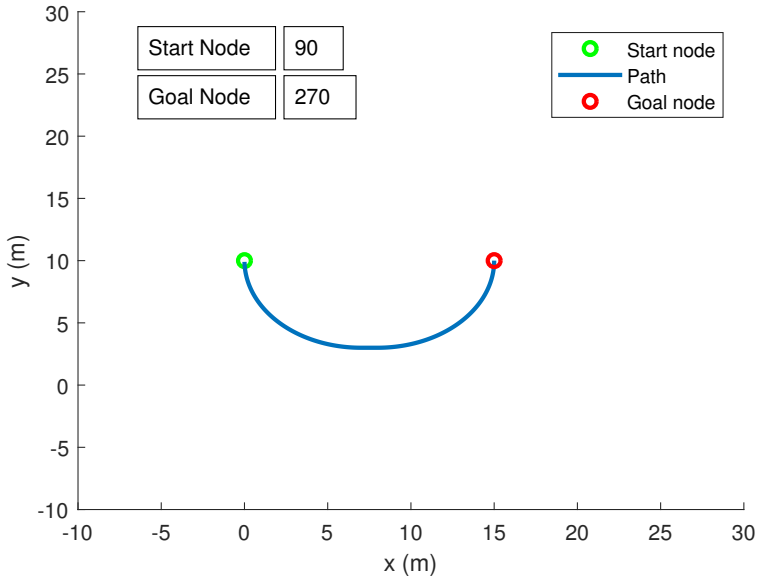

Figure 2.6: Illustration of Reeds-Shepp curve, showing the $L^-S^-L^-$ curve which is generated by the *CSC word* of the algorithm. The curve, depicted by the blue path, is generated with an initial angle of 90° and goal angle of 270°.

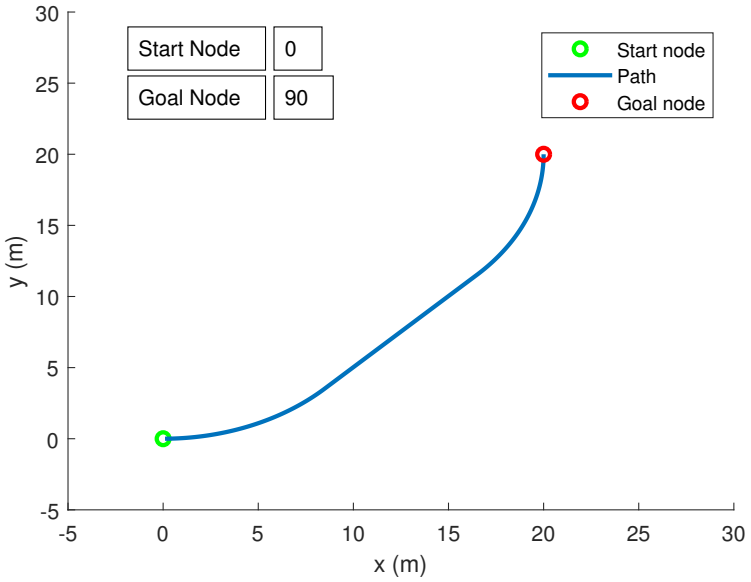

Figure 2.7: Illustration of Reeds-Shepp curve, showing the $L^+S^+L^+$ curve which is generated by the *CSC word* of the algorithm. The curve, depicted by the blue path, is generated with an initial angle of 0° and goal angle of 90°.

A few examples of the Reeds-Shepp curves can be seen in Figure 2.6 to Figure 2.9. It is observed that, based on the initial and the final angles, a short path is obtained that obeys the curvature constraint of the assumed vehicle.
It is not necessary to explicitly obtain formulae for all the different *words*. By making use of different transformation properties it is possible to obtain curves in

different directions. Using the "timeflip" transform it is possible to interchange all the '+' with '-'. Therefore, a *word* of form *CSC* having segments $L^-S^+L^-$ will have time-flipped segments of $L^+S^-L^+$. Thus, a point moving from (0,0,0) to (x,y,θ), *for example* in the reverse direction, will go from (0,0,0) to (-x,y,-θ), in the forward direction.

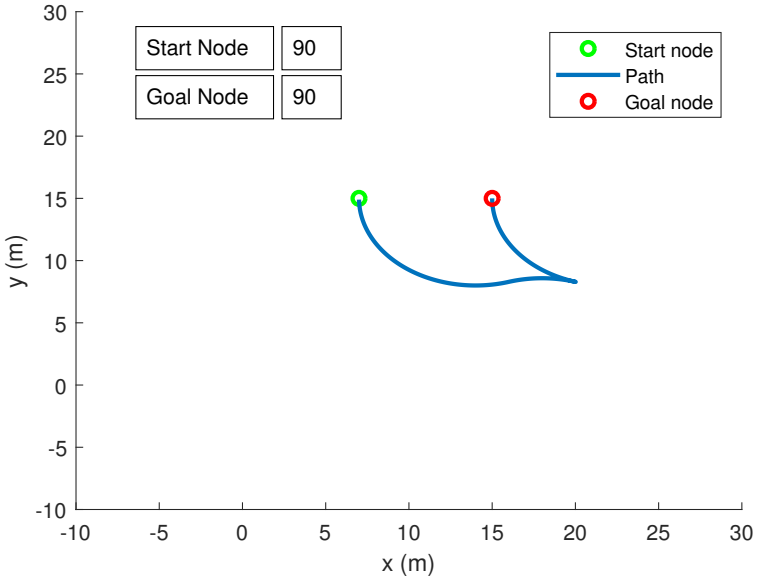

Figure 2.8: Illustration of Reeds-Shepp curve, showing the $R^-L^-R^+$ curve which is generated by the *CCC word* of the algorithm. The curve, depicted by the blue path, is generated with an initial and goal angles of 90°.

Similarly, a curve $R^+S^+R^+$ can be transformed to $L^+S^+L^+$ by interchanging 'L' with 'R' and vice verse. This property is referred as "reflect" transform. By this transformation the reflected path changes from (x,y,θ) to (x,-y,-θ). The "backward" transform can be used to execute the curve in reverse order, *i.e.* $L^+R^+L^-$ will transform to $L^-R^+L^+$, having (x,y,θ) transformed to ($x\cos\theta$+$y\sin\theta$, $x\sin\theta$-$y\cos\theta$, θ). Thus, by using these properties many redundant calculations can be eliminated.

An incremental search algorithm was also used by 'Talos', the autonomous vehicle by the MIT-team that finished in fourth place in DARPA Urban Challenge (2007), as mentioned in [3]. The team used RRT algorithm, [22], later optimised to RRT*. One of the disadvantages of this algorithm was the computational time. The jagged paths and efficiency had a trade-off with the number of samples, seen in [1].

The winner of DARPA Urban Challenge (2007), Boss [4] from The Stanford University, used a second-order spline profile for on-road navigation and a state lattice generation together with Anytime-D* for path planning in static environments, such as parking lots or off-road navigation. The second-order spline generated trajectories to follow the right lane and avoid obstacles along the road. The state lattice, [5], generated possible states that a vehicle could reach which included (x,y) coordinates and orientation of the vehicle. A path was generated on the state lattice

using Anytime-D* algorithm to perform the manoeuvre. A switching of these two algorithms was done to avoid the time to search for possible paths using the spline approach in unstructured environments.

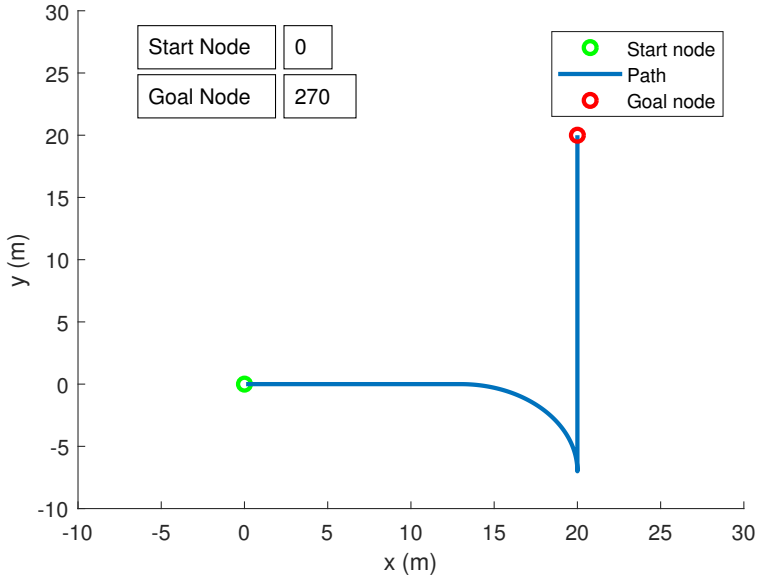

Figure 2.9: Illustration of Reeds-Shepp curve, showing the $S^+R^+S^-$ curve which is generated by the *SCS word* of the algorithm. The curve, depicted by the blue path, is generated with an initial angle of 0° and goal angle of 270°.

Some most common but critical aspects to be analysed, before deciding to take-up a certain algorithm for path planning, are the minimum safety distance from obstacles, feasibility of the path, computational time. When information regarding a path is given to an autonomous vehicle, it is important to generate paths to the goal from the current position such that the paths avoid obstacle collisions.

2.5 State lattice

The approach followed for navigation of the vehicle in this thesis is a modified Conformal Spatiotemporal method, which uses State Lattice as a foundation, seen in [6]. A state lattice is generally a grid on a map, consisting of nodes that can be traversed by a vehicle, an example can be seen in Figure 2.10. Feasible arcs respecting the non-holonomicity are generated between each node. The method in this thesis has the same concept, but is adapted to a structured road environment, illustrated in Figure 2.11, which will be further explained in Section 3.1. This method includes both path planning as well as trajectory generation to generate feasible trajectories by sampling future trajectories and numerically picking the best calculated trajectory based on a cost function. Paths are generated to states in front of the vehicle to a particular horizon. Along with it, the trajectories are generated.

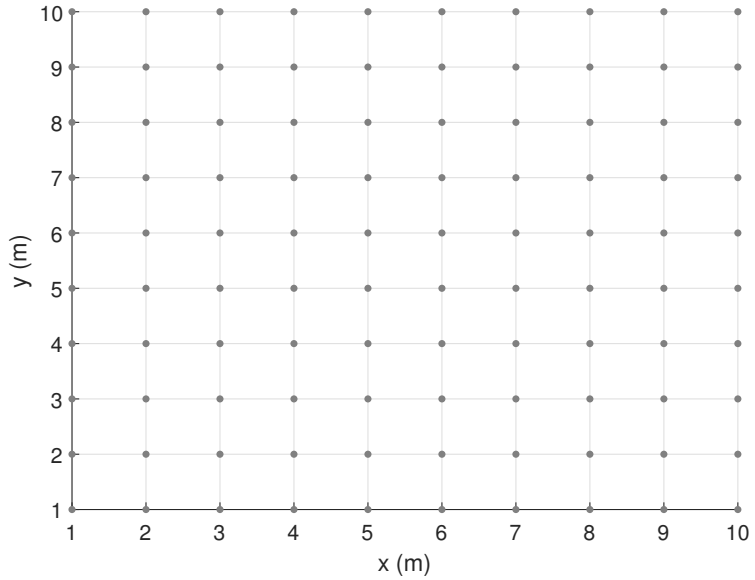

Figure 2.10: Illustration of a State lattice for an unstructured environment, showing a grid in which a vehicle can traverse.

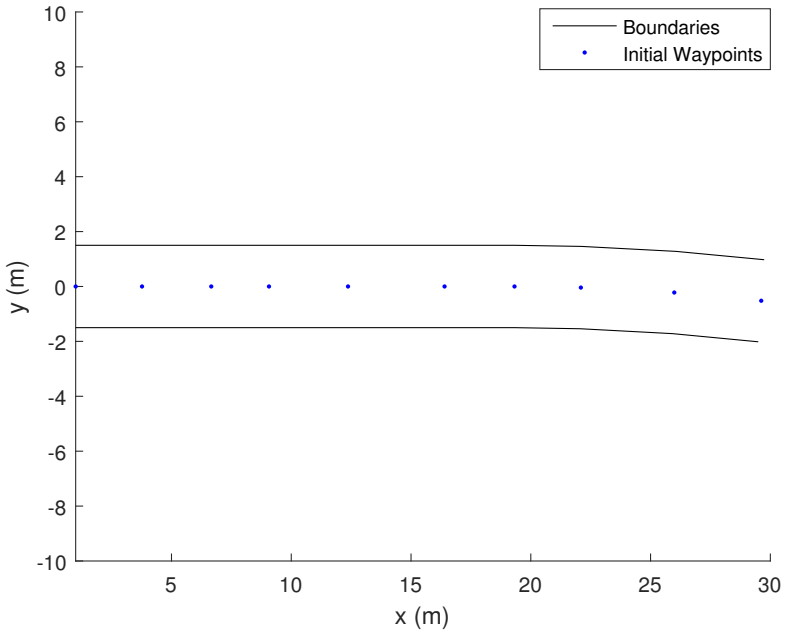

Figure 2.11: Illustration of a State Lattice approach for a structured environment, showing waypoints on a road with one lane. The road may also contain sign boards, which has to be considered while the autonomous vehicle manoeuvres to a goal position. Along the X-axis is the longitudinal road and the Y-axis is lateral road.

2.6 Hybrid-A*

The approach followed for parking the vehicle in this thesis is a version of the A* algorithm combined with Reeds-Shepp curve generation. This algorithm takes up the

path planning from the point where the vehicle enters an unstructured environment Figure 2.5 (parking depot or a parking lot), explained more in Section 3.2. The vehicle navigates through the waypoints and performs reverse parking, if required, once it is near the parking spot. The points to follow are extracted from the A* algorithm in an offline unstructured environment. The paths are generated in the simulation environment by using Reeds-Shepp curves for the vehicle to follow. A simple bicycle model is utilised to calculate the wheel angle from the orientation of the vehicle obtained from the Reeds-Shepp curves. The reverse curves are generated by the Reeds-Shepp model based on the final orientation required.

3

Method of Path Planning and Trajectory Generation

In this chapter, the methods applied for generating paths and trajectories are presented in depth. The first method presented is targeting the structured road environment, which consists of a speed limit and obstacles. The second method is targeting unstructured environments including obstacles. As the idea is to merge the algorithms so that it can demonstrate trajectory generations in valet parking scenarios, the algorithms switch at a pre-defined waypoint from the method in Section 3.1 to the method in Section 3.2. The Section 3.1 is inspired by Conformed State Lattice, [7], and the Section 3.2 is inspired by Hybrid-A*, [16]. The offline validation of the algorithms is done in using Matlab plots and the online implementation is done in PreScan simulation environment.

3.1 Structured Road Method

The idea behind this approach is inspired from [7] and [8], that are State Lattice based. The concept is realised by sampling a range of trajectories to future states of the vehicle as a function of time. The trajectories are numerically evaluated in a cost-function both for the generated geometrical paths as well as the speed profiles applied on the paths. The best trajectory is chosen accordingly.
The inputs to this algorithm are the current position, (x_r, y_r), current heading angle, θ, current velocity, v_r, the coordinates of the waypoints, (x_{w_i}, y_{w_i}), and an object detection camera providing the distance and Doppler velocity for the obstacles. In practice, a radar would be better to use regarding Doppler velocity. These inputs are assumed to reflect the ground truth in this project, *i.e.* the sensors are ideal. Given these inputs, cubic polynomials are generated to certain sets of waypoints in front of the vehicle, where these polynomials represent paths. Applying speed profiles on the paths, they are transformed into trajectories consisting of various parameters such as duration of trajectory, acceleration, jerk and centripetal acceleration. The trajectories are numerically evaluated based on numerous parameters. The most suitable trajectory is chosen with respect to safety, efficiency and comfort and output from the trajectory generator. This procedure is repeated for every iteration. The desired velocity is fed through a motion controller, consecutively converted to throttle percentage and brake pressure by a given conversion model. The curvature is based on the arc length whereas it is integrated to find the heading, (x, y) coordinates according to the bicycle model presented in Equation (2.8).

The algorithm is pre-evaluated in MATLAB, tested and evaluated in the real-time simulation environment PreScan/Simulink.

3.1.1 Architecture of Online Structured Road Method

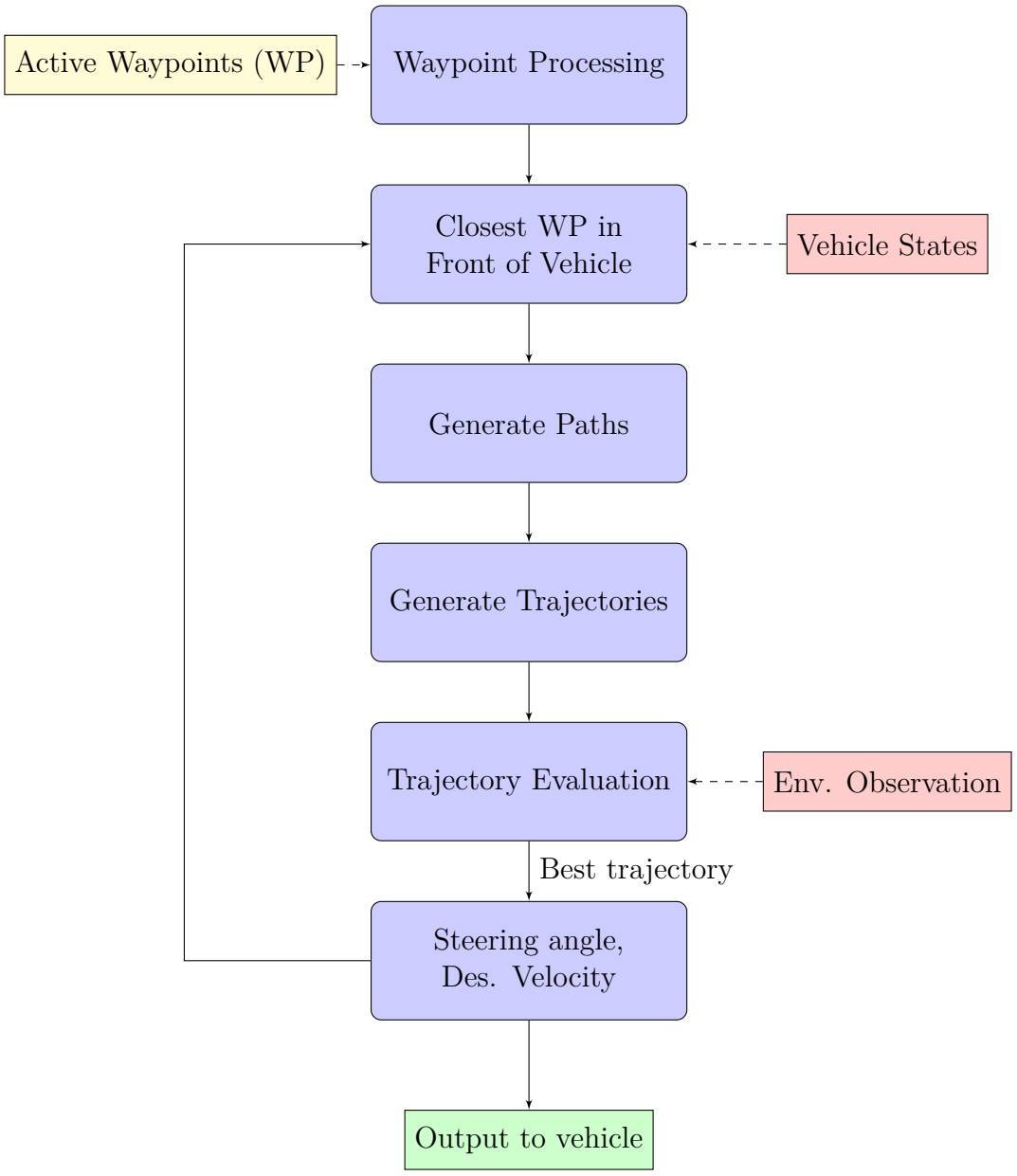

Figure 3.1: Flowchart of actions of the structured road method. The yellow box represents one pulse of input, in terms of given waypoints, to the algorithm. The initial action block, 'Waypoint Processing', approximates the states for every waypoint, addressing the heading and curvature at each waypoint, and increases the number of waypoints laterally along the lane. By receiving the states of the waypoints, the closest longitudinal point from the vehicle, along the road, is extracted. This is followed by path generation from the initial position, to each waypoint neighbour laterally. The red boxes represent frequently received data from the sensors, such as detected obstacles or vehicle states. Trajectories are applied and evaluated whereas the most suitable one is chosen according to a fixed cost function. The wheel angle and desired velocity are output from the algorithm. The green box represents the output.

The procedure of the algorithm for navigation in a structured road environment is seen in Figure 3.1. While running in a real-time environment, the algorithm follows a top-down approach and the sensor data is received continuously. The waypoints are given and assumed to be on the centre of the desired lane to follow. They are only computed once, in order to get the heading, curvature. If desired, the number of waypoints can be laterally increased. The paths are generated using cubic polynomials that are fit to the next set of waypoints in front of the vehicle, where the minimum distance to the waypoints is greater than a certain distance. The polynomials are converged to these set of waypoints using Newton-Raphson method, seen in Section 3.1.1.3, followed by speed profile application. Subsequently evaluating the trajectories based on a cost function that numerically computes the cost for the generated paths and speed profile applications, the best trajectory is chosen. The best trajectory's characteristics are output to the next layer, the motion control.

3.1.1.1 Waypoint Processing

Defining the given waypoints as

$$\eta_i = [x_{w_i} \; y_{w_i}] \tag{3.1}$$

where $i = 1, 2, ..., N$. N is the amount of waypoints given in the map, η_i represents the i^{th} waypoint including the (x, y) coordinate for the waypoints, denoted as (x_w, y_w). The distance between each consecutive waypoint is approximately 2 metres and the heading angle and the curvature for each point are approximated as

$$\theta_{w_i} = \tan^{-1}\left(\frac{y_{w_{i+1}} - y_{w_i}}{x_{w_{i+1}} - x_{w_i}}\right), \tag{3.2}$$

$$\kappa_{w_i} = \frac{\theta_{w_{i+1}} - \theta_{w_i}}{E_i}. \tag{3.3}$$

E_i represents the euclidean distance between η_i and η_{i+1} as the curvature is the derivative of the heading with respect to the arc length. θ_{w_i} and κ_{w_i} represent the heading angle and curvature of the i^{th} waypoint, respectively.
Extending Equation (3.1) to

$$\eta_i = [x_{w_i} \; y_{w_i} \; \theta_{w_i} \; \kappa_{w_i}], \tag{3.4}$$

each waypoint now consists of four states. In Figure 3.2, given waypoints are illustrated longitudinally along a one-way road, where each point includes the coordinates, the heading angle (orientation) and the curvature.
By adding a variable, d, defining the lateral offset from the sampled waypoints, an increased number of waypoints are sampled along the road, as in [8], written as

$$x_w(i, d) = x_{w_i} + d\cos\left(\theta_{w_i} + \frac{\pi}{2}\right), \tag{3.5a}$$

$$y_w(i, d) = y_{w_i} + d\sin\left(\theta_{w_i} + \frac{\pi}{2}\right), \tag{3.5b}$$

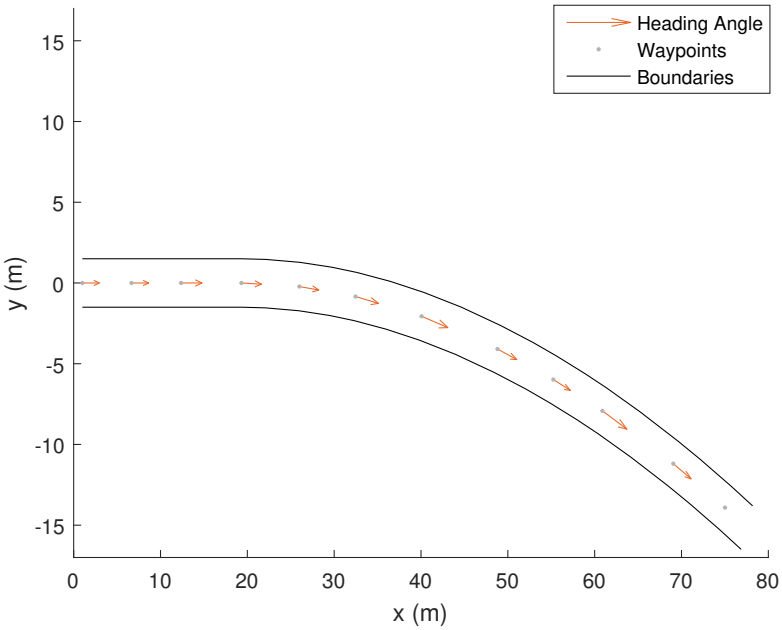

Figure 3.2: Illustration of waypoints given along a one-way road, starting from the left-hand side. The solid, black lines represent the road boundaries of a one-way road, the orange arrows represent the heading angle of each point and the grey points represent the given waypoints along a road, *i.e.* the stations.

$$\theta_w(i,d) = \theta_{w_i}, \tag{3.5c}$$

$$\kappa_w(i,d) = (\kappa_{w_i}^{-1} - d)^{-1}, \tag{3.5d}$$

illustrated in Figure 3.3. This augmentation requires information about the width of the road. Here, $(x_w, y_w, \theta_w, \kappa_w)$ represent the coordinates, heading angle and curvature of the i^{th} waypoint and the d^{th} lateral offset, respectively. For clarity, let i be called station, representing the longitudinal point along the road. Let the laterally increased waypoints at a station be called vertices, represented as d. The final vector is presented as

$$\eta_{w_{i,d}} = [x_{w_{i,d}}\ y_{w_{i,d}}\ \theta_{w_{i,d}}\ \kappa_{w_{i,d}}]. \tag{3.6}$$

3.1.1.2 Path Planner

As in [7], the path is defined as a cubic polynomial spiral, based on arc length, s, from current state to the goal state, implied as the desired waypoints. The current state of the vehicle, X_r, is defined as

$$X_r = [x_r\ y_r\ \theta_r\ \kappa_r], \tag{3.7}$$

where (x_r, y_r) are the coordinates of the ego-vehicle in the global Cartesian coordinate system. θ_r is the heading angle of the ego-vehicle and κ_r is the current curvature.

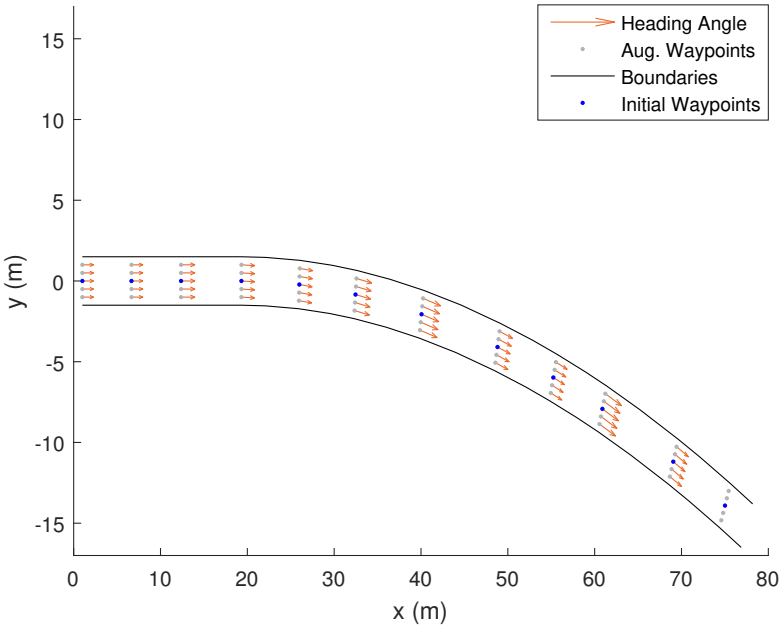

Figure 3.3: Illustration of increased number of vertices at each station along the road. The black, solid lines represent the road boundaries, the orange arrows represent the heading angle of each waypoint, the blue points represent the stations, *i.e.* the given waypoints. The grey points represent additional vertices at each station, *i.e.* the augmented waypoints at each station.

The target is to find a path connecting the current state X_r and a goal point $\eta_{w_{i,d}}$. A polynomial spiral representing the path is defined as

$$\kappa(s) = \kappa_0 + \kappa_1 s + \kappa_2 s^2 + \kappa_3 s^3, \tag{3.8}$$

in which arc length $s = 0$, starts from the vehicle's rear axle. κ is the curvature of a generated path, with respect to arc length, s. $(\kappa_0, \kappa_1, \kappa_2, \kappa_3)$ are the coefficients to be found. From Equation (3.8), the coefficients might vary largely in magnitude as s increases which might lead to round-off errors. In [7], a stable-path method is introduced where the coefficients in Equation (3.8) are parameterised with a vector P introduced as

$$P = [p_0 \; p_1 \; p_2 \; p_3 \; s_f], \tag{3.9}$$

with s_f as the total arc length of generated path. The variables from Equation (3.9) are defined as

$$p_0 = \kappa(0), \tag{3.10a}$$

$$p_1 = \kappa\Big(\frac{s_f}{3}\Big), \tag{3.10b}$$

$$p_2 = \kappa\Big(\frac{2s_f}{3}\Big), \tag{3.10c}$$

$$p_3 = \kappa(s_f), \tag{3.10d}$$

subsequently, the variables are treated to be of equal distribution along the path's arc length with κ representing the curvature throughout the generated path. The parameterisation yields the final definition as

$$\kappa(s) = \kappa_0(P) + \kappa_1(P)s + \kappa_2(P)s^2 + \kappa_3(P)s^3. \tag{3.11}$$

On solving Equation (3.11) for $s = 0,\ s = \frac{sf}{3},\ s = \frac{2sf}{3},\ s = sf$, the vectorised coefficients are found to be

$$\kappa_0(P) = p_0, \tag{3.12a}$$

$$\kappa_1(P) = -\frac{11p_0 - 18p_1 + 9p_2 - 2p_3}{2s_f}, \tag{3.12b}$$

$$\kappa_2(P) = 9\frac{2p_0 - 5p_1 + 4p_2 - p_3}{2s_f^2}, \tag{3.12c}$$

$$\kappa_3(P) = -9\frac{p_0 - 3p_1 + 3p_2 - p_3}{2s_f^3}. \tag{3.12d}$$

Consequently, the known variables are p_0 and p_3 since these variables represent the initial curvature of the path, $\kappa(0)$, for a given initial state, and the curvature of the goal point, $\kappa_{w_{i,d}}$, respectively for a desired i and d. The unknown variables are $p_1,\ p_2,\ s_f$. Further, having the curvature along the arc referred from the bicycle model in 2.3, the heading and (x, y) coordinates are retrieved along the arc length by integrating the curvature. These are obtained from Equation (2.8a) to Equation (2.8d) and are expressed as

$$\kappa_P(s) = \kappa_0(P) + \kappa_1(P)s + \kappa_2(P)s^2 + \kappa_3(P)s^3, \tag{3.13a}$$

$$\theta_P(s) = \theta_r + \kappa_0(P)s + \kappa_1(P)s^2 + \kappa_2(P)s^3 + \kappa_3(P)s^4, \tag{3.13b}$$

$$x_P(s) = x_r + \int_0^s \cos[\theta_P(s)]ds, \tag{3.13c}$$

$$y_P(s) = y_r + \int_0^s \sin[\theta_P(s)]ds, \tag{3.13d}$$

where $(\kappa_P, \theta_P, x_P, y_P)$ represent the states of the generated path from the current states, along s. The variable P differs for different paths, yet chosen such that p_0 is the initial curvature of a path, p_3 is the curvature of a chosen waypoint and the remainders are unknown, initialised with a guess. The integration for x_P, y_P along the path are approximated using Simpson's rule, from Equation (2.2). With p_1, p_2, s_f as initial guesses, the problem converges to find the unknown parameters such that

$$\Delta\rho = \underbrace{[x_{w_{i,d}}\ y_{w_{i,d}}\ \theta_{w_{i,d}}]}_{goal} - \underbrace{[x_P(s_f)\ y_P(s_f)\ \theta_P(s_f)]}_{path-end} \approx 0, \tag{3.14}$$

for a chosen waypoint of i and d and where $\Big(x_P(s_f), y_P(s_f), \theta_P(s_f)\Big)$ starts from vehicle's initial or current states. Thereby, s_f represents the path length from the initial coordinates to the chosen waypoint. $\Delta\rho$ is the vector of differences between a chosen waypoint and the generated path endpoint vector, X_P, including the initial guesses.

With the number of unknowns equal to the number of boundary conditions, the problem is solved using Newton-Raphson method, presented in Section 3.1.1.3, where the goal is to fulfil Equation (3.14). The minimisation problem is formulated as

$$\begin{aligned} \min_{P,e} \quad & e \\ \text{s.t.} \quad & \\ e &\geq \Delta\rho \\ e &\geq -\Delta\rho \\ p_0 &= \kappa_r \\ p_3 &= \kappa_{w_{i,d}} \end{aligned} \tag{3.15}$$

where p_0 is either the curvature of the current vehicle position or of any waypoint that should connect a path to a desired waypoint, $\kappa_{w_{i,d}}$.
Conclusively, let a generated path be denoted as $\tau_p(s)$ connecting the initial states, $s = 0$, and goal states, $s = s_f$.

3.1.1.3 Newton-Raphson Solution

Newton-Raphson method, presented from Equation (2.3), is introduced as

$$P^{n+1} = P^n - J_{P^n}\Big(X_{P^n}(s_f)\Big)^{-1}\Delta\rho\ m, \tag{3.16}$$

which is used to solve the problem of converging a path from the current position to a chosen waypoint. The Jacobian, J_P, is found by forward differences as calculated in [9], using small perturbations on (p_1, p_2, s_f).
The goal is to find a path that converges to one chosen waypoints from the current position or any initial position, as seen in Figure 3.4. By the use of augmented waypoints, several paths are generated, Figure 3.5, and the vehicle has the possibility to avoid obstacles by choosing an appropriate candidate. In this case, each τ_p (path) has different coefficients, P, and different goal states. For the lateral and longitudinal distances of the chosen waypoints, even unfeasible paths would theoretically be generated. This can be overcome by introducing a threshold that removes paths which include curvature greater then the vehicle's limited wheel angle, see Equation (2.10).
The procedure of the Newton-Raphson method is the same as in [7] and [9]. The steps are as following:

$$\Delta\rho \leftarrow [x_{w_{i,d}}\ y_{w_{i,d}}\ \theta_{w_{i,d}}] - [x_P(s_f)\ y_P(s_f)\ \theta_P(s_f)], \tag{3.17a}$$

$$J_{P^n}\Big(X_{P^n}(s_f)\Big) \leftarrow \begin{bmatrix} \frac{\delta x_P(s_f)}{\delta p_1} & \cdots & \frac{\delta x_P(s_f)}{\delta s_f} \\ \vdots & \ddots & \vdots \\ \frac{\delta \theta_P(s_f)}{\delta p_1} & \cdots & \frac{\delta \theta_P(s_f)}{\delta s_f} \end{bmatrix}, \tag{3.17b}$$

$$\Delta\hat{P} \leftarrow -J_{P^n}\Big(X_{P^n}(s_f)\Big)^{-1}\Delta\rho, \tag{3.17c}$$

$$P^{n+1} \leftarrow P^n + \Delta\hat{P}, \tag{3.17d}$$

where i, d are integers. Equation (3.17a) to Equation (3.17d) are repeated until $\Delta\rho \approx 0$.

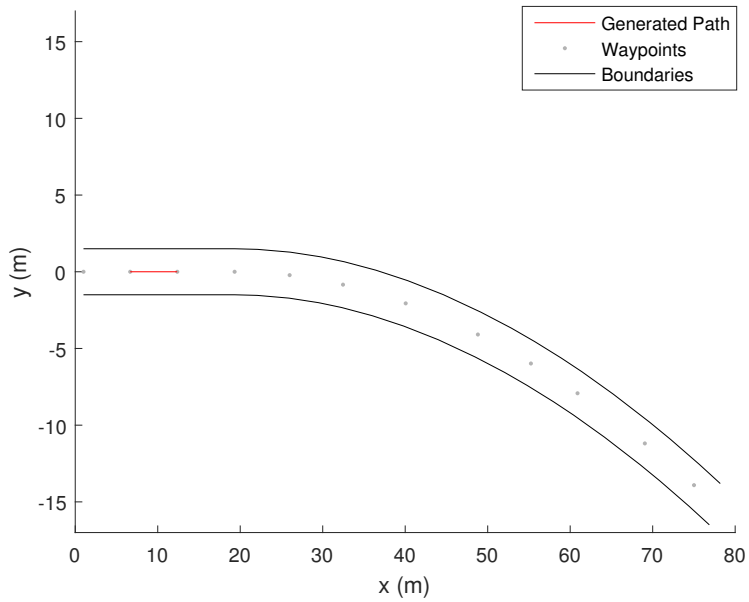

Figure 3.4: Illustration of one generated path, τ_p. The red line represents the path (cubic polynomial) after (p_1, p_2, s_f) have converged to fit the polynomial at station 3 from station 2. The first waypoint on the left-hand side represents station 1. The black, solid lines represent the road boundaries and the grey points represent the given waypoints.

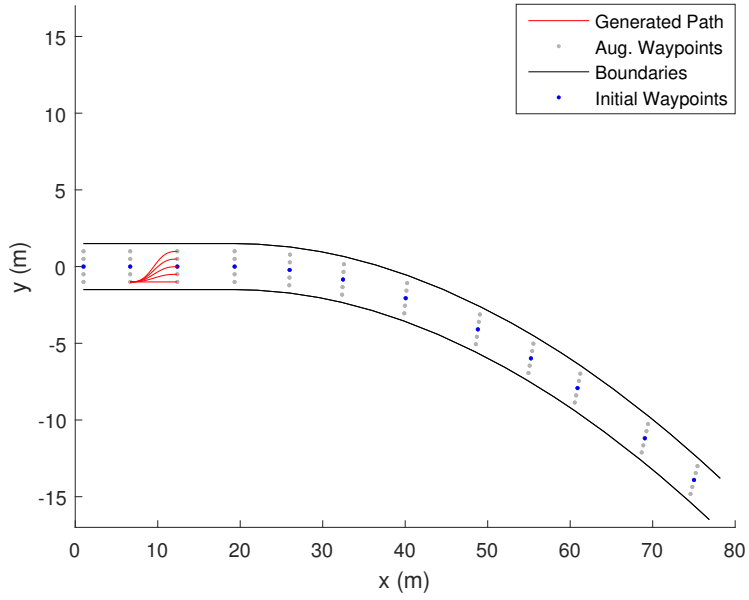

Figure 3.5: Illustration of generated paths, a set of $\{\tau_p\}$, after (p_1, p_2, s_f) have converged for each vertex at station 3 from the lowest vertex of station 2. The blue points represent the given waypoints, the grey points represent the augmented waypoints. The generated paths do not consider any steering limits.

3.1.1.4 Speed Generation

After generating geometric paths based on arc length, to turn the paths into trajectories (paths parameterised by time), a set of velocity profiles are applied to each path. The approach of this thesis is based on the idea that a fixed number of velocity profiles are applied to the set of all paths, $\{\tau_p\}$. The velocity profiles are generated as cubic polynomials to provide smooth acceleration, as in [8]. The trade-off of this approach is between computational time and accuracy. If more velocity profiles are generated, there are more velocity profiles to choose from resulting in a more accurate selection. *For instance* turns, where centripetal acceleration is the point of interest, or deviation from speed limit, where velocity is the point of interest.

Consider the following

$$\sigma_{init} = (s_0,\ v_0,\ a_0), \tag{3.18a}$$

$$\sigma_{goal} = (s_1,\ v_1,\ a_1) \tag{3.18b}$$

where σ_{init} and σ_{goal} represents the initial states and desired states of any chosen waypoint, respectively, s_0 and s_1 represents the initial and final arc length of the vehicle, respectively. *For instance*, s_0 is the initial position of a path, τ_p, and s_1 is the arc length, s_f, of the same path reaching to a waypoint. v_0 and a_0 serve as the obtained velocity and acceleration at the current state. v_1 represents a range of different, final velocities of a given path to a waypoint. a_1 is the final acceleration of the path.

The velocity polynomial is defined as

$$v(s) = \beta_0 + \beta_1 s + \beta_2 s^2 + \beta_3 s^3 \tag{3.19}$$

where s represents the arc length and v represents the velocity. The polynomial is constrained by the known parameters (v_0, a_0, v_0, v_1). To calculate the coefficients of Equation (3.19), the arc length is set such that $s = 0$ and $s = s_f$. Coefficients are found to be

$$\beta_0 = v_0 \tag{3.20a}$$

$$\beta_1 = a_0 \tag{3.20b}$$

$$\beta_2 = -\frac{2a_0 + a_1}{s_1} - \frac{3v_0 - 3v_1}{s_1^2} \tag{3.20c}$$

$$\beta_3 = \frac{a_0 + a_1}{s_1^2} + \frac{2v_0 - 2v_1}{s_1^3}. \tag{3.20d}$$

To obtain the uniform distribution from minimum end-velocity to maximum end-velocity of a path, the different end-velocity profiles, v_{1_i}, are found as

$$v_{1_i} = v_{min} + \frac{v_{max} - v_{min}}{N_{profiles-1}} i,\ i = 0, 1, ..., N_{profiles} - 1. \tag{3.21}$$

v_{min} represents the lowest possible end-velocity and v_{max} represents the highest possible end-velocity. $N_{profiles}$ represents the amount of distributed profiles to be generated. End-velocity implies the last velocity value to be reached at the end of the path.

In Figure 3.6, 10 different velocity profiles are applied to each τ_p from Figure 3.5, distributed equally between 0 ms^{-1} up to 10 ms^{-1} end-velocities. The paths are converted into trajectories, denoted as $\tau_t(s)$. Each τ_t consists of the corresponding path's states including a velocity profile. The acceleration and jerk are obtained from the velocity as

$$a = \frac{dv}{ds}\frac{ds}{dt} = (\beta_1 + 2\beta_2 s + 3\beta_3 s^2)v, \tag{3.22}$$

$$J = \frac{da}{ds}\frac{ds}{dt} = (2\beta_2 + 6\beta_3 s)v^2, \tag{3.23}$$

where a is the acceleration, J is the Jerk. v is the derivative of s with respect to time, also written as $\dot{s}$. The velocities represent the speeds throughout the trajectories with the direction along the paths.

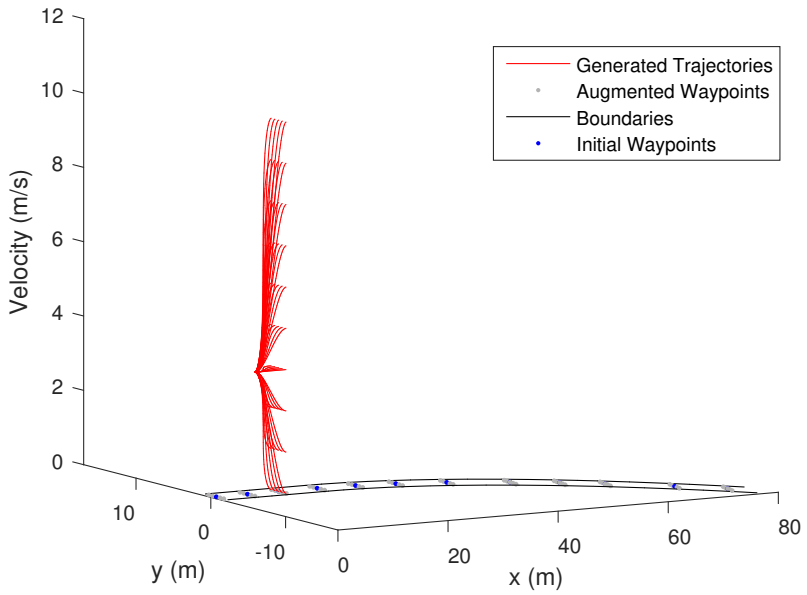

Figure 3.6: Illustration of applied velocity profiles as cubic polynomials, to the set of all $\{\tau_p\}$. By applying velocity profiles on the paths, the paths turn into trajectories, seen in the red lines starting from station 2 generated to station 3. A trajectory is denoted as $\tau_t(s)$. Here, the initial velocity is 3 ms^{-1} and the final velocities are equally distributed between 0 ms^{-1} to 10 ms^{-1}. Some trajectories may be unfeasible. The Z-axis represents the velocity, the black lines represent the road boundaries.

3.1.1.5 Cost Functions

Cost functions are a critical choice of the desired behaviour of an autonomous vehicle. Crucial factors in this field are safety, efficiency, comfort, where the key is to find a good balance that satisfies the stochastic nature.

Influenced by [8], the cost function is constituted by the parameters in Table 3.1, which target the geometric paths. The parameters in Table 3.2 target the time-parameterised paths, *i.e.* the dynamic cost.

Table 3.1: Cost function parameters for generated paths

PATH COST		
Parameter	**Mathematical Expression**	**Symbol**
Path length	s_1	C_l
Curvature	$\sum_{k=0}^{n} \lvert\kappa_k\rvert$	C_c
Curvature rate	$\sum_{k=0}^{n} \lvert\dot{\kappa}_k\rvert$	C_{cr}
Lateral offset from mid point	o	C_{lat}
Static obstacle coord.	$\sum_{k=0}^{n} \lvert d_{p_k}\rvert^2$	C_{dp}

Based on Table 3.1, n points are extracted from a trajectory, τ_t, to represent the cost, s_1 is the path length for a chosen τ_t. κ_k and $\dot{\kappa}_k$ are the curvature and curvature rate, respectively, for each extracted point, k. o is the euclidean distance between $\tau_t(s_f)$ and the corresponding middle-vertex of that station. o is calculated as

$$o = \Big|\Big|[x_P(s_f) - x_{wi,d}\ y_P(s_f) - y_{wi,d}]\Big|\Big|. \tag{3.24}$$

In Table 3.1, "*Static obstacle coord.*" represents an exponential function of euclidean distance from each extracted point of a trajectory to static obstacles on the road. d_{p_k} is calculated as

$$d_{p_k} = e^{-\frac{||[x_P(k)\ \ y_P(k)]^T - [x_{obs}\ \ y_{obs}]^T||}{\psi_p}}, \tag{3.25}$$

where $x_P(k)$ and $y_P(k)$ is extracted points from a trajectory. x_{obs} and y_{obs} is an obstacle location and ψ_p represents a bandwidth. Using this setup, the obstacle distance is formed as an exponential function where the value increases as the euclidean distance decreases. The functional criteria is formulated such that, if the distance to an obstacle is closer than a certain threshold, the cost is set to infinity.

In Table 3.2, t is the duration of a trajectory, v_g represents the end-velocity of a trajectory and v_{ref} represents the reference velocity on the road, *i.e.* the speed limit. The velocity, acceleration and jerk are represented by v, a and j respectively. The expression d_t represents the difference between the end-velocity of a trajectory and the Doppler velocity to an obstacle, computed with an object detection camera mounted on the vehicle, only for online purposes. The difference, d_t, is calculated as

$$d_t = (v_g - \phi)^2, \tag{3.26}$$

where ϕ is the Doppler velocity to the closest obstacle, d_s is an exponential function of the euclidean distance to an obstacle. The function, d_s, is calculated as

$$d_{s_i} = e^{-\frac{||[x_P(i)\ \ y_P(i)]^T - [x_{obs}\ \ y_{obs}]^T||}{\psi_t}}, \tag{3.27a}$$

where ψ_t represents the bandwidth. In this thesis, the bandwidth is set to be the end velocity of a trajectory. Adding a threshold as, if

$$\exists i \in \tau_t, ||[x_P(i)\ y_P(i)]^T - [x_{obs}\ y_{obs}]^T|| < t_{distance} \tag{3.28}$$

holds, then the weight for C_{ds} is greatly increased. The variable $t_{distance}$ represents a threshold value of maximum distance allowed. Using this parameter, if any of the extracted trajectory points are closer than $t_{distance}$, the cheapest outcome is chosen, resulting in a trajectory with a low speed.

Both acceleration and jerk are limited. By adding a threshold to these parameters, the cost is set to infinity if the limit is violated.

Table 3.2: Cost function parameters for the dynamic cost

DYNAMIC COST		
Parameter	**Mathematical expression**	**Symbol**
Duration	t	C_{dur}
Speed limit deviation	$\sum_{k=0}^{n}(v_g - v_{ref})^2$	C_s
Acceleration	$\sum_{k=0}^{n} a_k^2$	C_a
Jerk	$\sum_{k=0}^{n} j_k^2$	C_j
Centripetal acceleration	$\sum_{k=0}^{n} v_k^2 \kappa_k$	C_{ca}
Transformed distance to object	d_t	C_{dt}
Static obstacle coord.	$\sum_{=0}^{n} \lvert d_{s_k} \rvert^2$	C_{ds}

From Table 3.1 and Table 3.2, the total cost is calculated using the following equations,

$$C_{path} = w_l C_l + \frac{w_c C_c + w_{cr} C_{cr} + w_{lat} C_{lat} + w_{dp} C_{dp}}{n}, \tag{3.29a}$$

$$C_{Traj} = w_{dur} C_{dur} + \frac{w_s C_s + w_a C_a + w_j C_j + w_{ca} C_{ca} + w_{dt} C_{dt} + w_{ds} C_{ds}}{n}, \tag{3.29b}$$

$$C_{total} = C_{path} + C_{traj}, \tag{3.29c}$$

where C_{path} is the total of Table 3.1, C_{traj} is the total of Table 3.2 and C_{total} is the sum of these costs. The w's represents weights for each parameter where a higher weight implies more influence on the cost.

3.1.1.6 Trajectory Application

Each trajectory is compared and evaluated to each other through the cost functions. Path generation to 5 different waypoints along with 10 different speed profiles results in $10 \times 5 = 50$ different possibilities of trajectories. The cheapest trajectory is chosen.

From Equation (2.8d), depending on the architecture of the system, the curvature can be transformed to wheel angle, as in this project. The concept implies that the closest waypoint further than a fixed threshold in front of the vehicle is chosen, whereas the trajectories are generated to the given waypoint along with each of the laterally augmented waypoints at every instance, hence the cheapest among them is chosen.
Once trajectories have been generated the waypoints in the next station closest to the vehicle, the best one is chosen. By advancing the trajectory generation from that station to the next upcoming station and its vertices, the horizon is extended. This results in extra computational time yet more information of the forthcoming future states. The use of longer horizon can offer significant performance benefits in terms of smoothness, comfort, safety and efficiency.

Another way to extend the horizon is, except from generating to the forthcoming station, to also generate trajectories to the 2^{nd} and 3^{rd} forthcoming station, from the current position. A dynamic programming algorithm then compares the cheapest way to go to the station by comparing all possible trajectories from the current position. This method results in more smoothness, efficiency, comfort and safety. By repeating this pattern to a desired horizon, as the end state of each trajectory acts as the next initial state, an adaptive model-predictive planner is achieved.

There are many different ways of using the outputs from the trajectory generator. As in Equation (2.8d) of Section 2.3, the wheel angle of the best trajectory is, in this thesis, calculated after the curvatures of the best trajectory have been found. The trajectory states are the outputs containing positions, headings, curvatures and velocities.

3.1.2 Real-Time Configuration

To visualise the results of the Trajectory generation in PreScan and Simulink, a solution to track the generated trajectories has to be found. With the given waypoints and by obtaining the ego-vehicle's (x, y) coordinates, heading angle, velocity and information from the object detection camera, the algorithm in this thesis gives the curvature and desired velocity of the trajectory, at every iteration in the simulation environment. The curvature of the trajectory is stored in a vector representing the cubic polynomial, whereas the first elements are used to calculate the desired steering angle. The steering angle is fed through a PID-controller together with a low-pass filter to avoid fast changes. A saturation block limits the wheel angle to exceed the wheel angle limit of the vehicle.
The data from the object detection camera is processed such that the closest obstacle in front of the ego vehicle has its Doppler velocity measured. The output velocity from the algorithm is regulated with a PID-controller together with a low-pass filter and saturation. The output from PID-controller is converted via a given conversion model, to desired throttle and braking for the ego-vehicle. Due to the cost function presented in Section 3.1.1.5, the ego vehicle is able to adapt its velocity to obstacles detected by the camera by either adjusting to the velocity differences or by braking to reach a stationary state, further elaborated in Section 4.3.

3.2 Unstructured Environment Method

This section is focused towards the second part of the thesis, *i.e.* to perform the parking action once the vehicle is off the structured road. In the field of autonomous vehicles, unstructured or semi-structured environments represent the map for navigation of the vehicle in which there are neither structured lanes, that provide a set of waypoints, here mentioned as path points, along the path to the goal, nor sign boards that indicate specific road instructions. The unstructured environment is usually generated based on sample-based methods, which gives control over the scaling of the search space, [14]. The most common unstructured environments, as mentioned in [15], are parking lots, cross-sections in Urban environments, deserts. According to the same article, an initial offline path is obtained on a map of this environment. During this, the vicinity of obstacles is also included. Different algorithms can be followed for generation of paths in these types of environments. The most common path search algorithms are observed to be A*, D* or RRT*. Inspired by an application for parking in an unknown environment by Stanford Racing Team's vehicle at DARPA Urban Challenge, the search algorithm chosen for this part is similar to Hybrid-A*, [16]. The algorithm is executed in Matlab/SIMULINK followed by the simulation in PreScan.

3.2.1 Architecture of Unstructured Environment Method

In general, the offline path to be followed is obtained by using the modified A* algorithm, which is generated using the inputs (x_r, y_r, θ_r) and (x_g, y_g, θ_g). x_r, y_r, θ_r are the initial and x_g, y_g, θ_g are the final global coordinates and vehicle orientation (heading) in degrees. Points are extracted from the A* path based on a selective distance between them, which depends on the turning radius of the vehicle. The continuous Reeds-Shepp curves are generated to the extracted points, consecutively, from the current position of the vehicle. The acceleration to the vehicle is varied corresponding to the velocity throughout the manoeuvre. The control of the vehicle is updated online in the simulation by varying the wheel angle and acceleration, which are the outputs obtained in this thesis, with respect to the current position and orientation of the vehicle. A flow chart explaining the algorithm can be seen in Figure 3.7. Further, this section explains the path following algorithm for unstructured environment in detail.

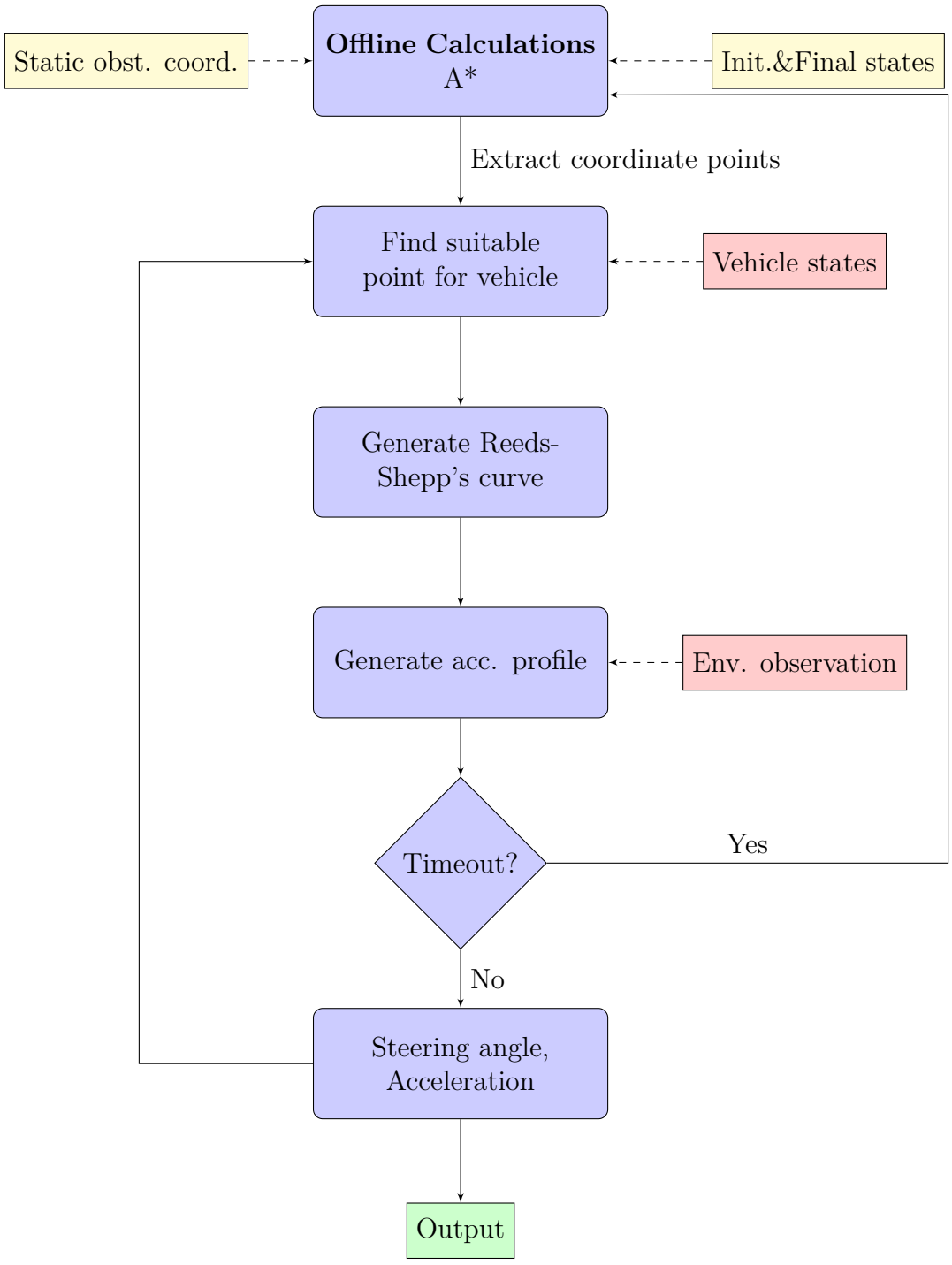

Figure 3.7: Flowchart of Unstructured Environment Method. The yellow boxes represent the inputs at the initial instance. The **Offline Calculation**, mentioned in the top blue box, is an A* algorithm. Path points are extracted from the A* path. Information regarding the vehicle state and the environment, depicted by red boxes, are provided at every instance during the simulation. Following this, the Reeds-Shepp curves are generated to suitable path points based on the current position. The acceleration profile of the vehicle is adjusted by setting a threshold velocity for the vehicle. Replanning of the **Offline Calculation** could be done if the online trajectory generation times-out, as seen in the decision block 'Timeout'. The obtained outputs are wheel angle and acceleration, which are given to the controller of the vehicle, seen as the green box.

3.2.2 Inputs and Outputs

A set of inputs are required for calculating the path and generating trajectories. The turning radius of the vehicle shall be decided based on the scenario. The extracted path points are similar to the waypoints in Structured Road Method.
The required inputs for offline path calculation are:

- The initial and final global coordinates of the vehicle, (x_r, y_r) and (x_g, y_g),
- The initial and final orientation of the vehicle, θ_r and θ_g,
- The turning radius of the vehicle,
- The obstacle coordinates.

The inputs for online path generation are:

- The current position and orientation of the vehicle at each instance, $(x_{r_o}, y_{r_o}, \theta_{r_o})$,
- The path points, $(x_{w_i}, y_{w_i}, \theta_{w_i})$,
- The turning radius of the vehicle,
- The threshold velocity.

The maximum curvature, κ_{max} of the vehicle is calculated from the turning radius as seen in Equation (2.8d). A depiction of an environment with initial points, final points and obstacle coordinates can be seen in Figure 3.8.
The output to the vehicle's controller is acceleration, a, and wheel angle, δ, calculated based on the curvatures and heading angles obtained from the Reeds-Shepp curve generation.

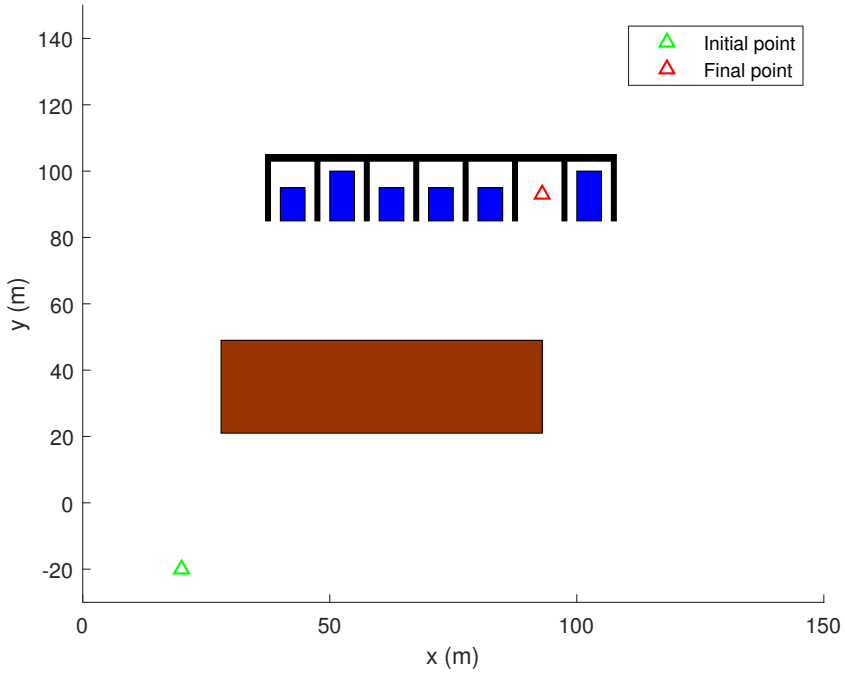

Figure 3.8: Illustration of an unstructured environment, showing obstacles. The green and red markers represent the initial and final positions between which the path is to be generated. The blue and brown boxes represent obstacles in the environment.

3.2.3 Hybrid A-star

Typically, in an unstructured environment there are nodes which define a path. They are scattered over the map and selected ones can be activated to reach the goal. Thus, to decide which ones to activate and follow might become a difficult decision task. To avoid this complex case, a short path is generated to the goal, beforehand. Therefore, offline calculations are performed to obtain the reference points to be followed. Hence, only a few points are chosen from these coordinates based on a threshold distance. These points are referred as the path points. The path obtained by using the A* algorithm can be seen in Figure 3.9, where straight lines are drawn between the chosen nodes of a grid, similar to Figure 2.2. As seen, the drawback is that the obtained path from the A* algorithm is holonomic, which means that it is explicitly non-traversable by a vehicle. A hybrid version of this algorithm is formulated, such that the obtained shortest path also includes approximations of the orientations throughout each node along the path. The path on the map represents the information about the position and orientation of the vehicle such that they are at a minimum distance away from any obstacle coordinate. The cost to a neighbour node is assigned from the current node as it moves towards the goal node. This is different from the normal A* method, in which the cost for every node is assigned based on the given map boundaries before the search starts.

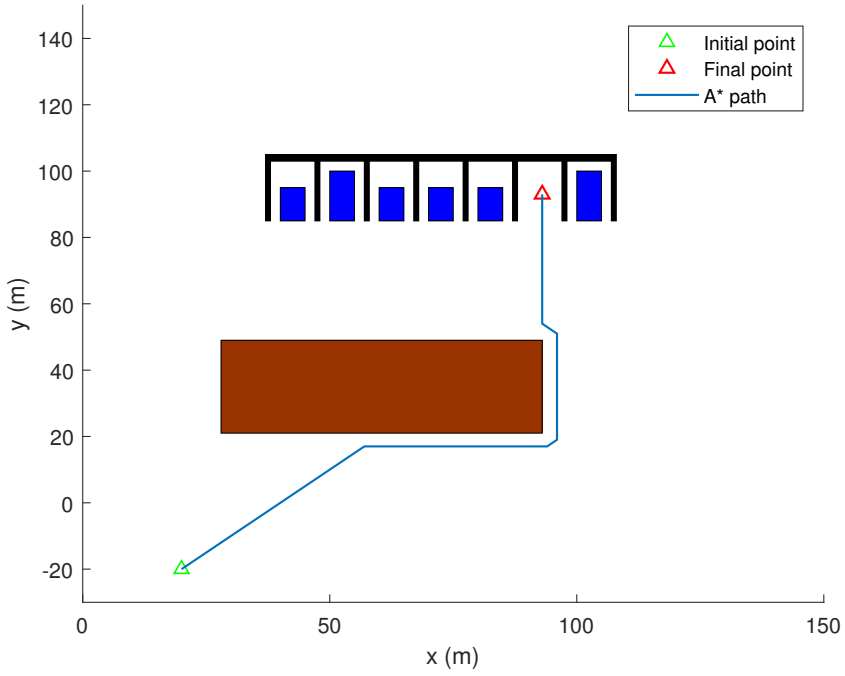

Figure 3.9: Illustration of the A* path, showing the shortest path to the goal by avoiding the obstacle (brown box). Depicted from this figure is that the generated path is holonomic and has sharp turns which are impossible to traverse by a vehicle.

If the map contains obstacles, a path is searched such that they are avoided. A safety distance is maintained such that there is a margin between every node of the

path and the obstacles. This is taken into consideration because this A* path is obtained offline and the extracted path points are given for online tracking of the vehicle, considering that the vehicle has a certain width and length.
The path obtained from this discrete-search based method is holonomic on the map. Each node comprises of an approximated angle which is the angle between the two consecutive points and does not reflect the nature of the A* path itself. The holonomic characteristic of the A* path is solved by the curve generation between path points by the Reeds-Shepp method. Before moving to curve generation by Reeds-Shepp method, path points have to be derived from the nodes of the A* path. These are similar to the waypoints in the Structured Road Method. They are chosen from the nodes of the A* path based on the distance between two consecutive path points, which should be greater than at least twice the length of the vehicle. This is to avoid unwanted and congested cusps (the point where the vehicle changes direction). Figure 3.10 shows the path points, containing the information about the direction to move from that node. The path points are denoted as $(x_{w_i}, y_{w_i}, \theta_{w_i})$ where the i^{th} node is characterised with (x, y) coordinates and orientation.

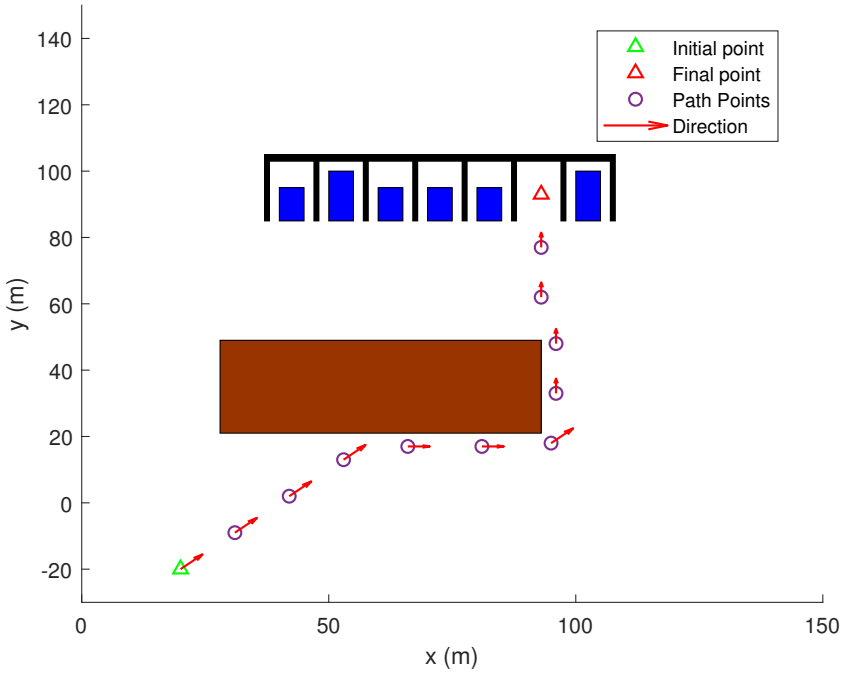

Figure 3.10: Plot illustrating the information obtained from the A* path, containing the angles of each path point. The purple markers are the path points the are extracted from the A* path. The direction of the nodes, on each path point, is pointed by the red arrows. Each path point carries information about the position and orientation (heading) of the node from the A* path. The green and red markers refer to the initial and final positions.

The above method of extracting path points from the A* path works well for forward parking scenarios, since the vehicle has to manoeuvre only in one direction. In a case where the vehicle has to reverse-park, this method of extracting the path points

might result in undesired curves. Hence, to overcome this problem a sub-point has to be defined. During the offline calculation, an A* path is first generated to a sub-point, seen in Figure 3.11. The sub-point is included in the path finding algorithm. Initially, the path is generated to the sub-point, following which an extended path from the sub-point to the goal is achieved. The sub point is generated such that it is placed at a certain distance below the parking spot next to the desired parking spot, on the opposite side of the entry direction of the path. The angle is chosen to be perpendicular to the goal angle, which is outwards from the parking spot, and faced away from the entry of the path. *For example*, if the vehicle comes from the left, the point is placed below the parking spot to the right of the desired parking spot, facing rightwards, and vice-verse.

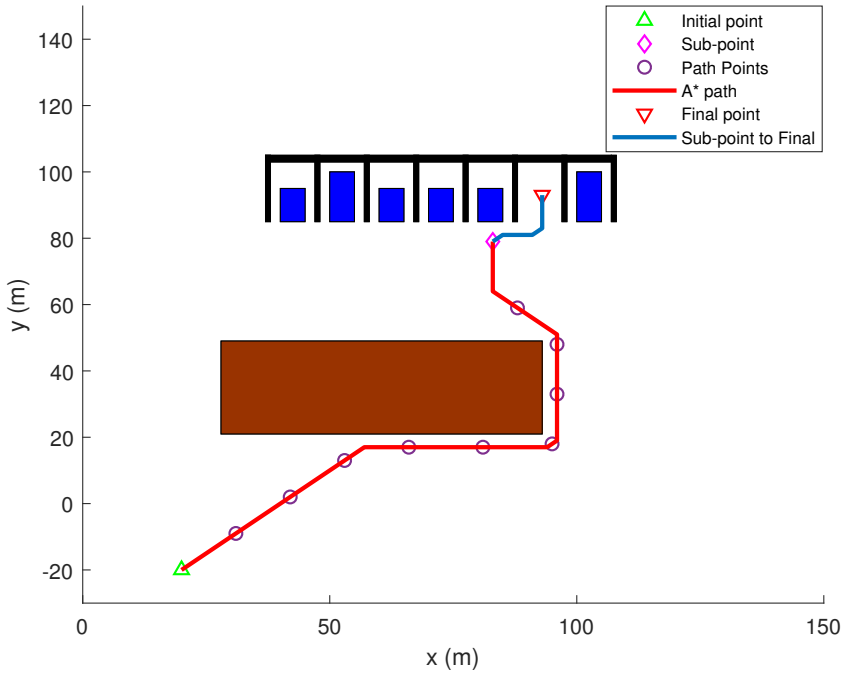

Figure 3.11: Plot illustrating the A* path to the sub-point and further extended from the sub-point to the goal. The path points extracted from the total A* path are the purple markers on the A* path. The red dashed line represents the A* path to the sub-point and the blue line is the extended path. The diamond shaped magenta marker indicates the sub-point. Here, the sub-point is placed on the left side of the desired parking spot since the entry direction of the vehicle is seen to be from the right side.

3.2.4 Reeds-Shepp Curve Generation

Once the points to be followed are extracted, a non-holonomic path is generated from the current position to a certain horizon of retrieved path points. These paths are generated by Reeds-Shepp curves.

Reeds-Shepp method is adapted to generate a curve to the closest path point, as the vehicle moves towards the final position. The curves are generated based on the calculation of the Reeds-Shepp parameters, as seen in Equation (2.20) to Equation (2.27). The curve is generated from the current position to the consecutive path point, as depicted in Figure 3.12. A constraint is set such that, if the closest subsequent path point is in a distance less than a certain value then the path point to follow is changed to the following look-up point (*i.e.* the next path point), as seen in Figure 3.13.

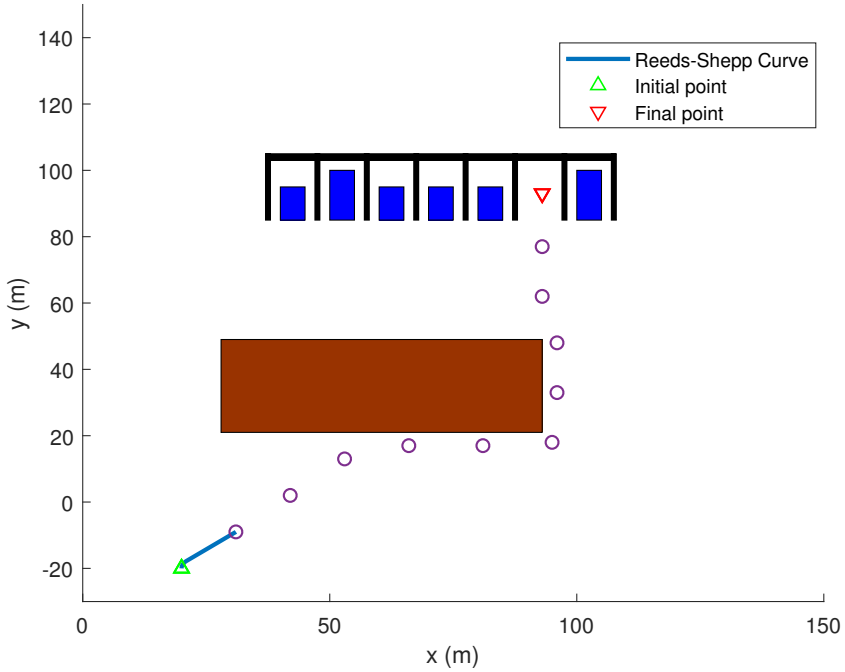

Figure 3.12: Illustration of a Reeds-Shepp curve generation (the blue line) to the next path point from the initial point. After generating the first curve to the forthcoming path point, the initial position for the next curve is set to be the attained path point. This would result in a continuous path from the vehicle's current position at any instance to the following path point and eventually to the goal.

To calculate the Reeds-Shepp parameters, to obtain a curve to the i^{th} path point, the vehicle states are transformed to the Spherical Coordinate System using the obtained (x_r, y_r, θ_r), at the vehicle's current state, and (x_{w_i}, y_{w_i,w_i}), which are the states at i^{th} path point for a chosen i. The transformation can be written as

$$x_{r_o} = \Delta x cos(\Delta\theta) + \Delta y sin(\Delta\theta), \tag{3.30}$$

$$y_{r_o} = \Delta y cos(\Delta\theta) - \Delta x sin(\Delta\theta), \tag{3.31}$$

$$\theta_{r_o} = \Delta\theta, \tag{3.32}$$

where

$$\Delta x = x_{w_i} - x_r, \tag{3.33}$$

$$\Delta y = y_{w_i} - y_r, \tag{3.34}$$

$$\Delta \theta = \theta_{w_i} - \theta_r, \tag{3.35}$$

in which x_{r_o}, y_{r_o} and θ_{r_o} are, hereby, the current vehicle coordinates in the local coordinate system at each instance. The parameters are used to calculate the different curves possible from the equations seen in (2.17) to (2.19). The shortest curve among the possible curves is chosen as the path.

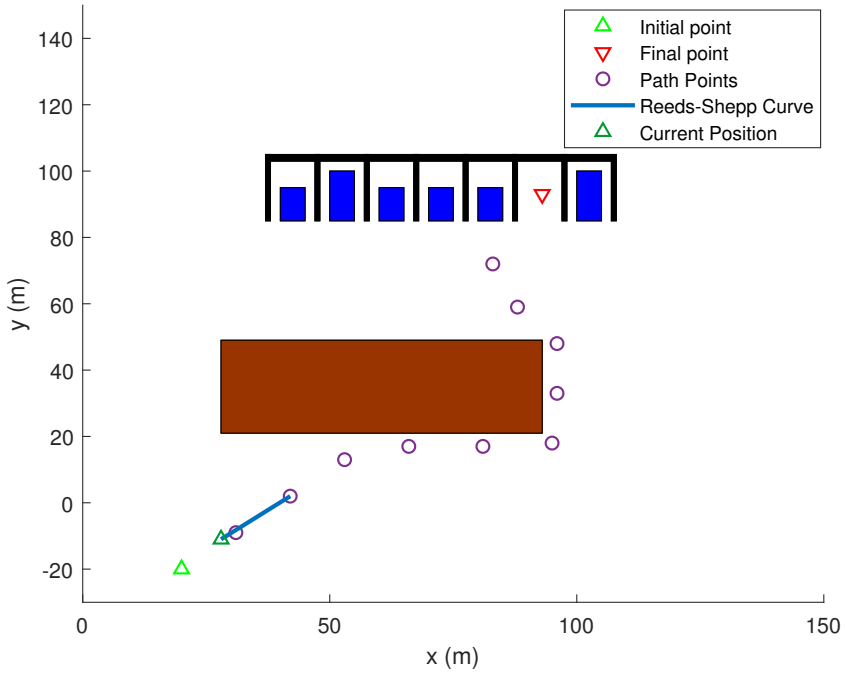

Figure 3.13: Plot showing the Reeds-Shepp curve generation (the blue line) to the following look-up path point, since the subsequent path point is closer than a specific distance. The dark green marker represents the position of the vehicle at an instance, assuming it is moving to the 2^{nd} path point but generates the curve to the 3^{rd} path point, since it is near the consecutive path point (*i.e.* 2^{nd} path point).

In Figure 3.13, the Reeds-Shepp curves can be generated between the path points, where the initial position must be changed to be the next path point consecutively, after a prior path has been generated. A non-holonomic path is generated from the current position to the goal. The problem of generation of Reeds-Shepp curves to the desired parking spot is that is does not obey the constraints of the obstacles. This can be overcome by the sub-point generation and further, path point extraction in the A* computations. To make this method work in an real-time configuration, Reeds-Shepp curves are iteratively generated to the closest path point which fulfils the criteria, having the initial position as the current position at every iteration. That means that the horizon of 1 path point is used.

3.2.5 Acceleration Profile and Wheel Angle

Since the environment is a parking lot or a similar open space, there is assumed to be no speed limit. Hence, lower velocities are sufficient for the manoeuvre. A low threshold value is set as a desired constant velocity. An acceleration profile is generated based on the velocity of the vehicle. If the current velocity goes below the threshold velocity, a small constant acceleration is given to the vehicle. If the velocity exceeds the threshold velocity, no acceleration is given. Hence, a velocity close to the threshold is maintained throughout the path. Since it is assumed that there is no speed-limit inside an open parking lot, it is safe to have a lower threshold velocity to perform the manoeuvre. It is necessary to change the gear for reversing the vehicle. Thus, a deceleration is given as the output so that the vehicle slows down and reaches a near stop position when the vehicle reaches the sub-point, to change from the current forward gear to reverse gear. A deceleration is also provided when the vehicle reaches the goal position so that the vehicle stops, irrespective of the current gear.

The desired wheel angle, which is also another output to the controller, is calculated by differentiating the heading along the path. Since the curvature is the derivative of the heading, with respect to the arc length, the curvatures along the path are obtained by

$$\kappa_P(k) = \frac{\theta_P(k+1) - \theta_P(k)}{\sqrt{\left(x_{w_P}(k+1) - x_{w_P}(k)\right)^2 + \left(y_{w_P}(k+1) - y_{w_P}(k)\right)^2}}, \tag{3.36}$$

where k represents the discretised step along the path, from $1, 2, ..., K$. Here, K indicates all the elements of the path. $\kappa_P(k)$ is the curvature of the path at the k^{th} step. The denominator in Equation (3.36) is the approximated arc length at each element. The curvature is translated to desired wheel angle by using Equation (2.8d).

4
Results

In this chapter, the finalised results along with values of different parameters are introduced. The first section, Section 4.1, showcases the achieved results of the on-road navigation offline. The second Section, 4.2 showcase the achieved results of the off-road navigation offline. The remainder of this chapter investigates the testing of the online environment simulations, for which the data is extracted from the simulations. Online results refer to the results achieved in a simulation.

4.1 Offline Structured Road Method

For the offline generation, given the initial waypoints and the road width, $road_w = 3$ metres, the augmented waypoints are generated using Equation (3.5a) to Equation (3.5d), where d is computed as

$$d = \Big[-\frac{road_w}{3},\ -\frac{road_w}{6},\ 0,\ \frac{road_w}{6},\ \frac{road_w}{3} \Big]. \tag{4.1}$$

As the waypoints are given such that the distance between each waypoint is 2 metres, every 2^{nd} station is filtered out since the distance would be too close otherwise. The initial guesses and the remaining parameters for every variable in P is initialised as

$$p_0 = 0, \tag{4.2a}$$
$$p_1 = 0, \tag{4.2b}$$
$$p_2 = 0, \tag{4.2c}$$
$$p_3 = 0, \tag{4.2d}$$
$$s_f = 4. \tag{4.2e}$$

These values were found to achieve good results with Newton-Raphson method. From Equation (3.21), the velocity configuration is set as

$$N_{profiles} = 10, \tag{4.3a}$$
$$v_{min} = 0, \tag{4.3b}$$
$$v_{max} = 10, \tag{4.3c}$$
$$a_0 = 0, \tag{4.3d}$$
$$a_1 = 0. \tag{4.3e}$$

4.1.1 Cost Function Configuration

The weights in the cost function referred in Table 3.1 and Table 3.2 of Section 3.1.1.5, are presented in Table 4.1. The values are found by trial and error where 'Path Cost' is found offline, in Matlab, and 'Dynamic Cost' is found online, in Simulink and PreScan. Apart from the weights in the cost function, a threshold is added to the acceleration. The acceleration threshold, $a_{threshold}$, is set to 2. A trajectory containing any element 'a', such that if

$$\exists i \in \tau_t,\ |a_i| > a_{threshold} \tag{4.4}$$

holds, acquires an increase of 10^4 units to the cost. To intervene emergency situations, if an obstacle is less than 5 metres in front of the vehicle, $a_{threshold}$ is rejected. The bandwidth, ψ_p in Equation (3.25), is set to $\psi_p = 1$.

Table 4.1: Weights used in Cost Function.

PATH COST		**DYNAMIC COST**	
Symbol	*Weight*	*Symbol*	*Weight*
w_l	1	w_{dur}	0.1
w_c	2	w_s	1
w_{cr}	1	w_a	0.5
w_{lat}	90	w_j	0.01
w_{dp}	10^{11}	w_{ca}	1.4
		w_{dt}	20
		w_{ds}	20

In Equation (3.28), $t_{distance} = 1.5$ metres. Due to the threshold, if Equation (3.28) holds, *i.e.* if a particular trajectory has any element less than 1.5 meters away from the obstacles, $w_{ds} = 10^{10}$ for that trajectory.

4.1.2 Plot Results

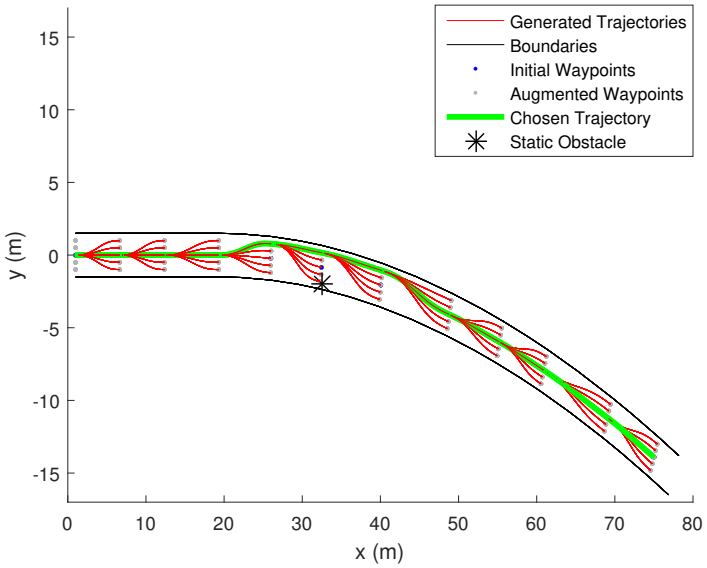

Figure 4.1: Illustration of the chosen offline generated trajectories (red lines) from the initial waypoint, station 1 in the left of the plot, to station 12. The best trajectory, to one station forward, is chosen to be used as the initial at next iteration. The black asterisk represents a static obstacle on the road. The trajectory avoids the obstacle and returns back to the centre of the lane. The horizon is 1 station.

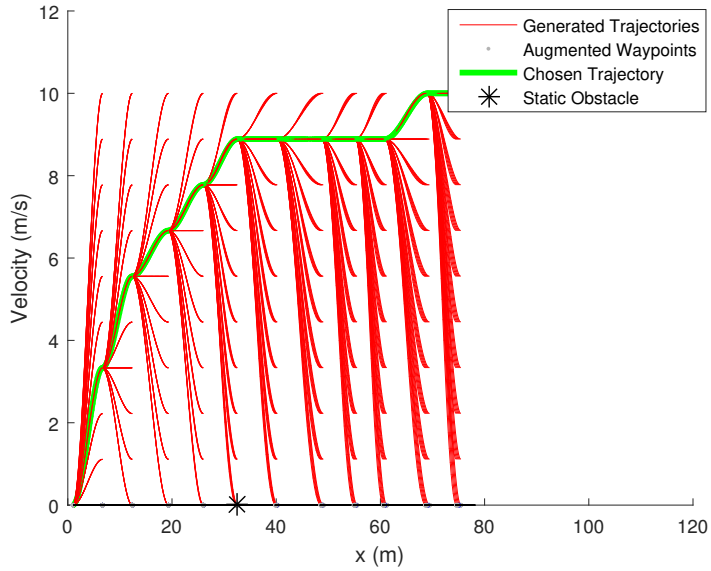

Figure 4.2: Illustration of the chosen, offline generated trajectories (red lines) from the initial waypoint, station 1 in the left of the plot, to station 12. The black asterisk represents an obstacle on the road. The Z-axis represents the velocity. In this figure, the acceleration is smoothed up to the desired velocity, set to $10ms^{-1}$.

In Figure 4.1 and Figure 4.2, the trajectories are generated to the next station in front of the current station. As the best trajectory has been chosen among the generated ones, the endpoint of that trajectory is assigned to be the next initial station, *i.e.* the horizon is 1 station. The reference speed is set to be $10ms^{-1}$. A static obstacle is included to be at $(x, y) = (32, -2)$, which is considered by the algorithm. Dynamic obstacles are not added to the offline generated trajectories. In these figures, every second station is removed to increase the distance of the trajectories.

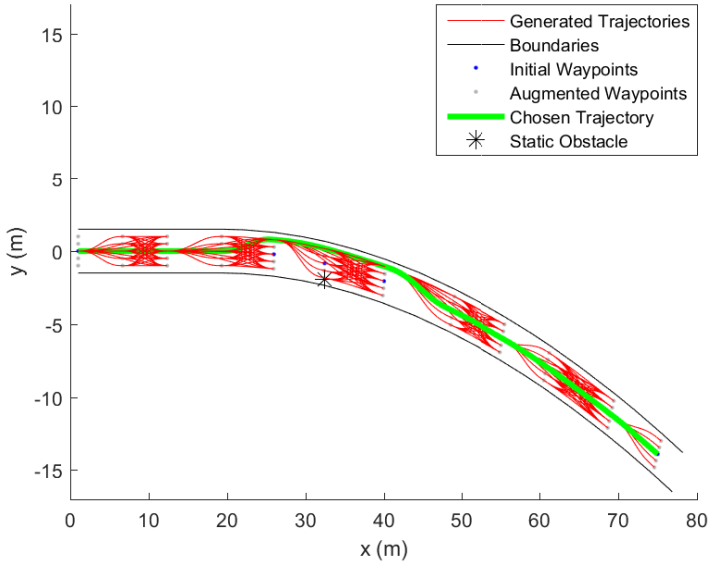

Figure 4.3: Illustration of the chosen, offline generated trajectories from the initial waypoint at station 1 in the left of the plot to station 12, where the best trajectory to 2 stations further is chosen and the end-point is used as the initial point at next iteration. The black asterisk represents a static obstacle on the road. The trajectory clearly avoids the obstacle and returns back to the centre of the lane again. The red lines represent the generated trajectories.

In Figure 4.3 and Figure 4.4, the horizon is extended to 2 stations, implying that the algorithm checks the numerically best possible trajectory segment from 1 station to 2 stations further away. This is done by either choosing a trajectory segment that goes straight to 2 station further away or by choosing a trajectory segment that goes from the initial station to the forthcoming station, and another trajectory segment from that station to the next coming station. In this case, there is no difference between the paths of Figure 4.1 and Figure 4.3. The clearest difference is in the acceleration profiles, seen between Figure 4.2 and Figure 4.4, where the acceleration profile is smoother in the latter figure. Another difference is the constant velocity that is obtained in Figure 4.2. The desired velocity of the whole plot is $10ms^{-1}$. Another big difference is the computational time. Computing trajectories to a horizon of 2 results in $(5\text{x}10) + (5\text{x}10) + (5\text{x}5\text{x}10) = 350$ different trajectories to evaluate in this case, compared to $(5\text{x}10) = 50$ trajectory as in Figure 4.1 and Figure 4.2. Another scenario is presented in Figure 4.5 and Figure 4.6, where an obstacle is

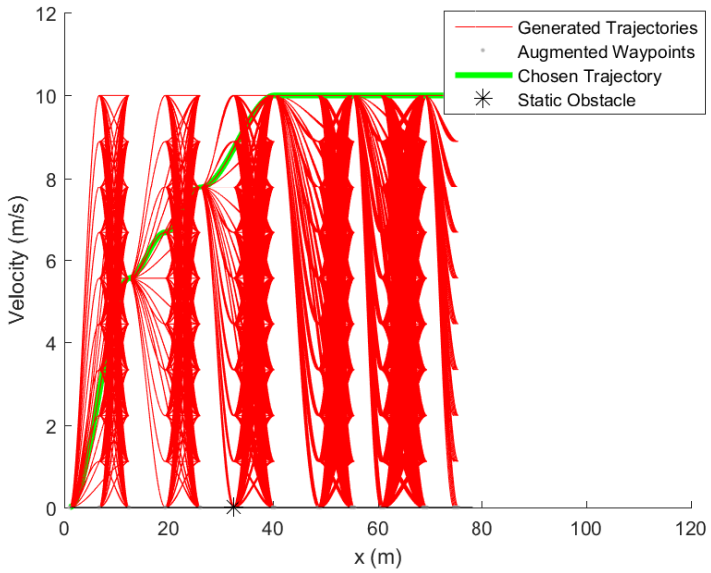

Figure 4.4: Illustration of the chosen, offline generated trajectories from the initial waypoint at station 1 in the left of the plot to station 12. The black asterisk represents an obstacle on the road. The Z-axis represents the velocity. The desired velocity is set to $10ms^{-1}$. A horizon of 2 stations is used.

placed in the middle of the road. As the road width is 3 metres and the threshold, $t_{distance}$ is 1.5 meters, the condition in Equation (3.28) will execute. Seen in Figure 4.5, by assuming that a vehicle is around 1.8 meters wide, the obstacle will have an impact on the vehicle. Hence, looking at Figure 4.6, the velocity is reduced to $0ms^{-1}$, resulting in a stationary state before the obstacle.
Every 4^{th} waypoint is used in Figure 4.5, to increase the distance between each station. The distance between each given waypoint is 2 metres, using every 4^{th} waypoint results in a minimum distance of approximately 8 metres for the trajectories. As some trajectories are generated straight to 2 stations further away and include the fact that the trajectories are arcs, the distance for some segments of a trajectory end up to be around 20 metres or more.
Instead, by using every 6^{th} waypoint when generating trajectories, seen in Figure 4.7, the distance can be set to be 12 metres between the initial waypoints. With trajectories generated to 2 station further away, length of some trajectory segments become 30 metres or more. By removing too many stations, as seen in Figure 4.7, the distance between each station becomes too big, resulting in trajectories surpassing the road boundaries.

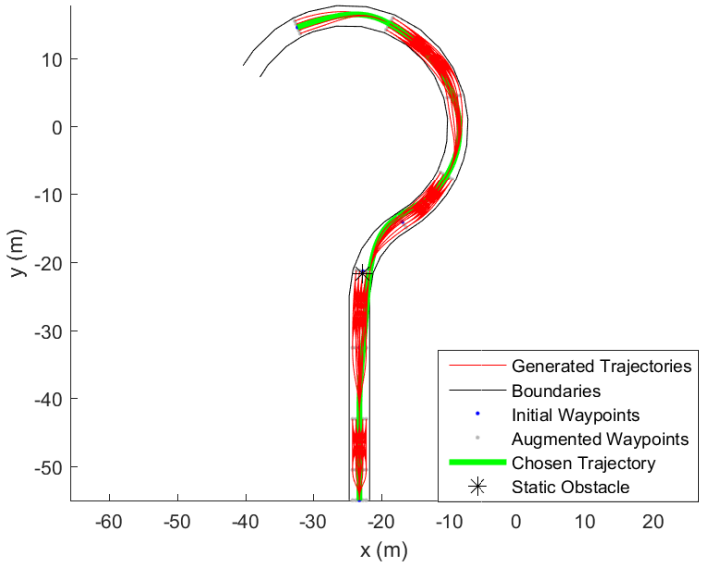

Figure 4.5: Illustration of the chosen, offline generated trajectories from the initial waypoint at station 1 in the bottom of the plot to station 10, upper left, in another scenario. Every 4^{th} station is included, which is another way to extend the distance of trajectories. The black asterisk represents an obstacle on the road. The station horizon is 2. In this case, an obstacle is placed in the middle of the road which the vehicle cannot avoid without adjusting the velocity.

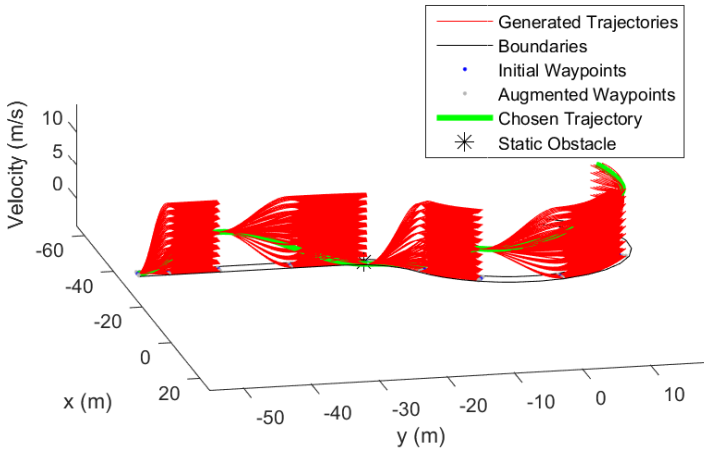

Figure 4.6: Illustration of the chosen, offline generated trajectories from the initial waypoint at station 1 in the bottom of the plot to station 10. The black asterisk represents an obstacle on the road. The Z-axis represents the velocity. The desired velocity is set to $10ms^{-1}$. A horizon of 2 stations is used. In this case, the vehicle exceeds the threshold limit, $t_{distance}$ and the planned velocity is reduced.

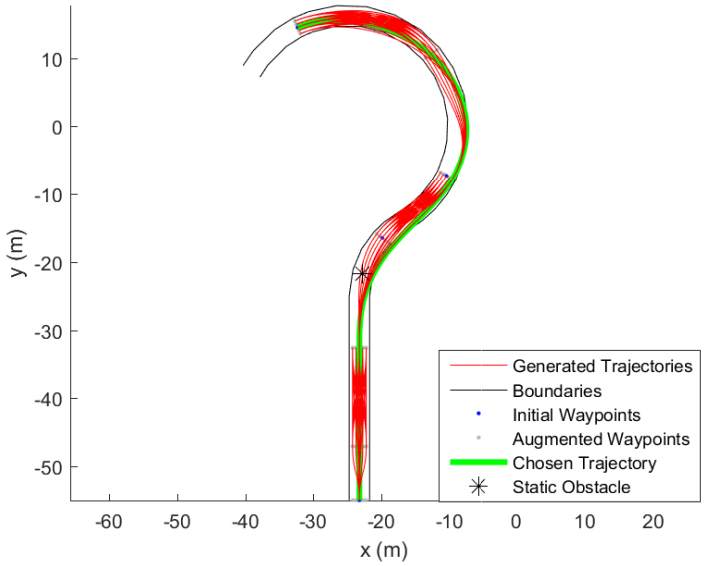

Figure 4.7: Illustration of the chosen, offline generated trajectories from the initial waypoint at station 1 in the bottom of the plot to station 7. The black asterisk represents an obstacle on the road. A horizon of 2 stations is used. In this case, the vehicle exceeds the threshold limit, $t_{distance}$ and the planned velocity is reduced. The planned trajectory stretches outside the road boundaries due to the long distance between each station.

4.2 Offline Unstructured Road Method

Once the set-point is reached, which indicates that the vehicle has entered the parking lot, the Unstructured Road Method takes over the path planning to the goal position. In this section, the values used for the test cases showcased are mentioned and the behaviour is illustrated.

4.2.1 Parameter Configuration

The parameters configured to implement parking of the vehicle are,

- The initial and final positions,
- Current vehicle angle and desired goal angle,
- Curvature of the path, based on the length of the vehicle,
- Safety distance.

The obstacles are avoided, with a certain distance to ensure safety. The map is provided along with obstacle coordinates. The position and angle of the sub-point (in case of reverse parking) is set in the A* algorithm to obtain obstacle free path points.

4.2.2 Test Scenario 1

Test Scenario 1, as depicted in Figure 4.8, illustrates a case in which a vehicle is required to park from an initial angle of 90° to a goal angle of 90°. In this case, a

sub-point generation is not required as the vehicle has to park in the same direction as its current orientation. It is seen that the safety distance is maintained from the obstacles, which is 10 metres for this case. The Table 4.2 specifies the information regarding the values of the parameters that are set for this case. Here, all the other parking spots, other than the goal, are assumed to be obstacles.

Table 4.2: Parameters for Test Scenario 1.

Parameter	*Value*	*Unit*
Initial Position	(15,0)	*m*
Final Position	(85,94)	*m*
Initial Angle	90	*degrees*
Final Angle	90	*degrees*
Length of the Vehicle	3	*m*
Safety Distance	10	*m*

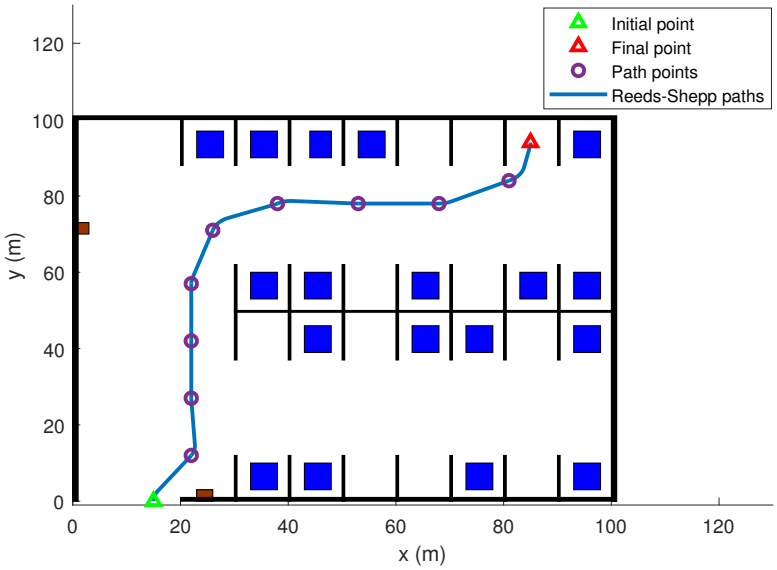

Figure 4.8: Illustration of Reeds-Shepp path, generated from an initial angle of 90° to a goal angle of 90°. The green and red triangle markers represent the initial and final positions, while the purple markers are the path points between which the Reeds-Shepp curves are generated. The Reeds-Shepp paths are generated based on the length vehicle which is 3 metres. A safety distance of 10 metres is maintained between the surrounding objects.

4.2.3 Test Scenario 2

Test Scenario 2, as depicted in Figure 4.9, illustrates the path generated by Reeds-Shepp algorithm. As seen in Table 4.3, based on the given input parameters, the A* path is generated to the goal point. It is observed that Reeds-Shepp algorithm provides a forward path from the initial point without generating a sub-point. Then a reverse curve is generated at the end of the path to park the vehicle in the desired orientation. This validates the Reeds-Shepp path, showing that the path generated between two points obeys the curvature of the vehicle's path. Therefore, it is important to extract path points at a reasonable distance between them, from the A* path. An important observation here is that the safety distance should be set in the A* algorithm with respect to the size of the vehicle.

Table 4.3: Parameters for Test Sccenario 2.

Parameter	*Value*	*Unit*
Initial Position	(0,85)	*m*
Final Position	(95,45)	*m*
Initial Angle	0	*degrees*
Final Angle	180	*degrees*
Length of the Vehicle	5	*m*
Safety Distance	2	*m*

4.2.4 Test Scenario 3

In this scenario, another case of reverse parking is seen. In Figure 4.10, the A* path is shown, which is generated to a sub-point and to the goal. This ensures a check for availability of obstacles while the Reeds-Shepp path is generated to the sub-point. From Figure 4.10, it is also seen that the Reeds-Shepp path does not exactly follow the A* path, which explains the difference between the paths generated by A* and Reeds-Shepp algorithms. While generating a path for the vehicle to follow, a condition is set such that, if the current position of the vehicle is within a range of 1 metre to a path point, the algorithm chooses the next path point to generate the path. This case is seen in this scenario and it is observed that the Reeds-Shepp algorithm handles it by generating the curve to the goal instead of trying to reach the sub-point. The parameters used for this test scenario are tabulated in Table 4.4.

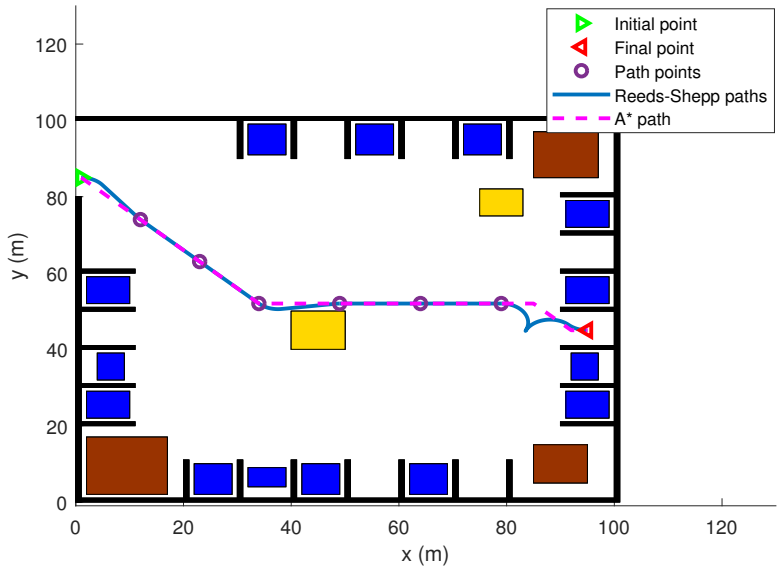

Figure 4.9: Illustration of Reeds-Shepp path along with the A* path. Apart from other parameters, the magenta coloured dotted lines represents the A* path. In this case the safety distance to the obstacles is set to 2 metres. The length of the vehicle is set to 5 metres.

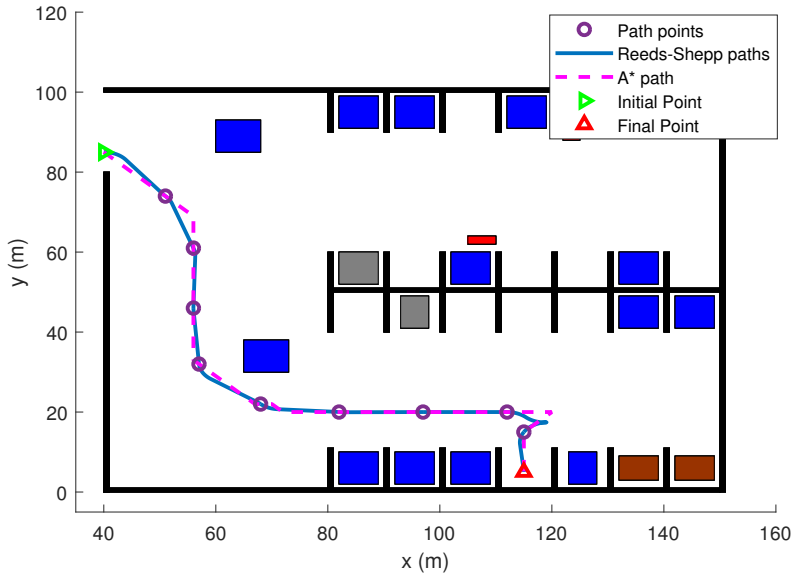

Figure 4.10: Illustration of Reeds-Shepp path on the A* path indicating the reversing action based on the sub-point generation from the A* path. An observation made from this is that the path points taken from the A* path along with the sub-point are chosen based on the distance mentioned while extracting the path points.

Table 4.4: Parameters for Test Scenario 3.

Parameter	*Value*	*Unit*
Initial Position	(40,85)	*m*
Final Position	(115,5)	*m*
Initial Angle	0	*degrees*
Final Angle	270	*degrees*
Length of the Vehicle	5	*m*
Safety Distance	4	*m*

4.3 Simulation Results

In this section, the algorithms are run online. In Figure 4.11, an illustration of the simulation environment is introduced and the same map is seen as a 2-D plot in Figure 4.12. A car is initialised on a structured road that reaches the parking area. using the structured road method. Once the car has reached the parking area, there is a transition to the unstructured road algorithm. A parking spot is given to the algorithm. Static obstacles are present in the parking lot whereas dynamic obstacles are present on the structured road, seen in Figure 4.11. The algorithms are implemented in an online configuration in PreScan. A bicycle model reflects the vehicle's dynamics in the simulation. Generated trajectories are extracted periodically from the simulation to illustrate the trajectories from the simulations, in a plot.

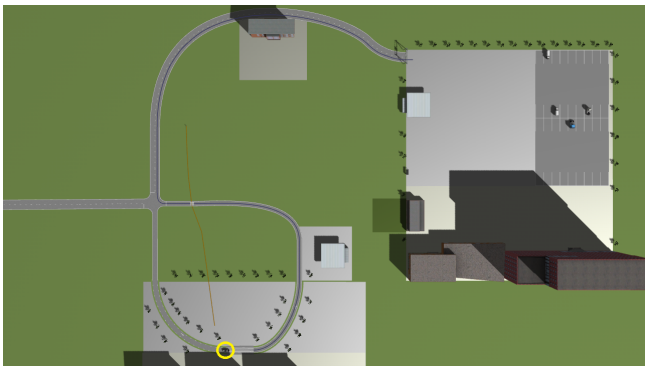

Figure 4.11: The map simulation environment run in online simulations, illustrating a valet-parking scenario. The ego-vehicle is in the lower end of the image, which is highlighted by a yellow circle. The blue line in front of the vehicle is an example trajectory desired for the whole path, reaching from the initial position to the parking area. The upper right open-space is the entrance to a parking lot. The darker area is the parking lot. The orange line in the centre of the image is the trajectory of a pedestrian crossing the road.

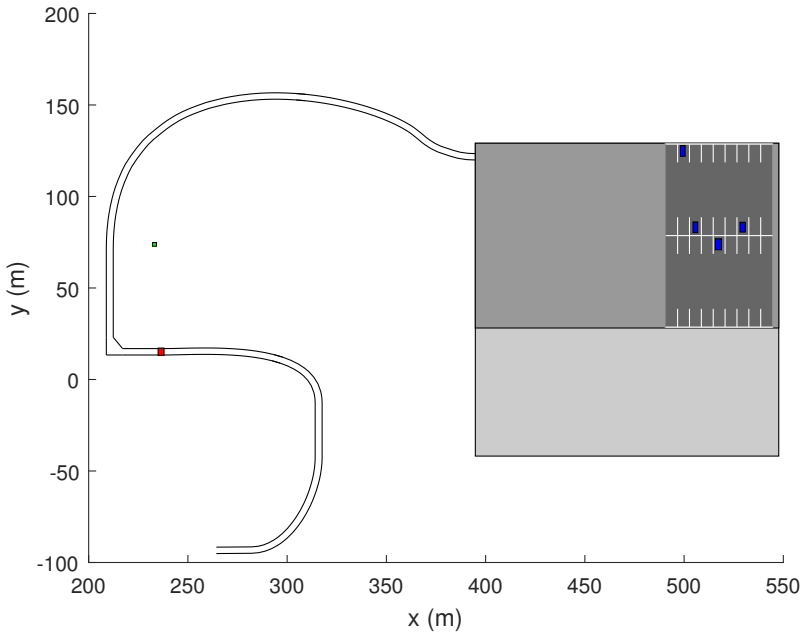

Figure 4.12: A simplified plot of the road and the parking area, in the simulation environment. The darkest grey square represents the parking lot, the white lines in the parking lot represent the parking spot alignments. The mid-grey square is the entry space from the road to the parking area. The initial position of the road is at $(260, -90)$. The red box is the pedestrian-crossing on the road, the green box is the pedestrian's initial position and the blue boxes are parked vehicles in the parking lot, representing static obstacles.

4.3.1 Online Structured Road Method

For the structured road, in comparison to the offline-generated results, the method is simplified due to the computational requirements when running online. The differences are presented below:

- only one d is used, *i.e.* only the initial waypoints are considered. The augmented waypoints are excluded.
- The horizon is 1, *i.e.* trajectories are only generated to the first waypoint in front of the vehicle, with a requirement of minimum distance of 3 metres away. The trajectories are generated from the current position at every iteration, meaning that the initial states are the ego-vehicle states at every iteration.
- The reference velocity is $6ms^{-1}$.
- Given the initial waypoints approximately every 2 metres apart, every other waypoint is removed as the distance would be too close to the vehicle. The benefit of having frequent waypoints is that the angle approximation gets better.
- 14 different velocity profiles are applied to the generated path, ranging from $0ms^{-1}$ to $8ms^{-1}$.
- The object detection camera only detects vehicles and humans. In this case, only the detections within a range of 25 metres are of interest, whereas the

Doppler velocity of the closest object is measured.

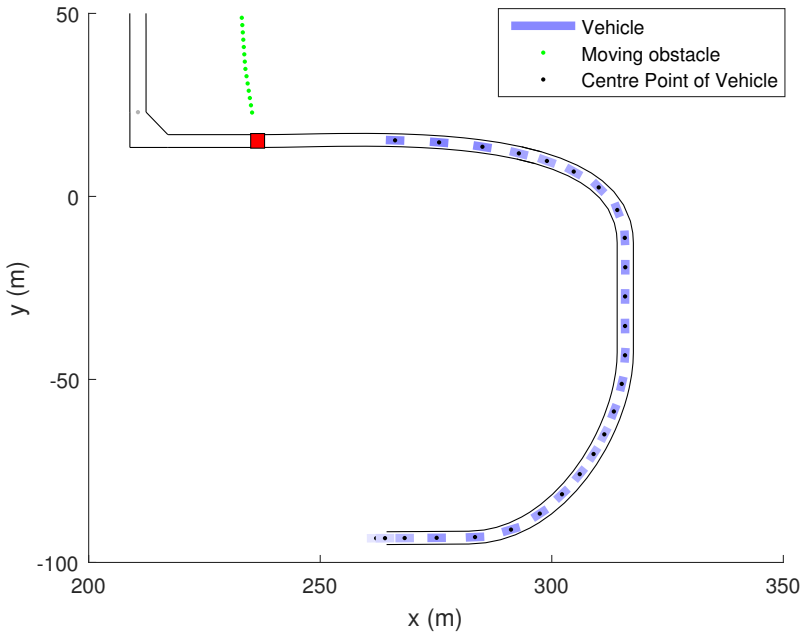

Figure 4.13: Illustration of the motion of the vehicle in the simulation environment. The green points represent a moving pedestrian. The pedestrian starts from the top and moves downwards during the simulation. The red box is the pedestrian-crossing on the road, the blue boxes represent the vehicle. The black points are the centre point of the vehicle, at every 40^{th} iteration. The length of the vehicle box is 4 metres. The frequency of the simulation is 20 Hz. The data is extracted for every 2 seconds of the simulation and the vehicle is initialised in the bottom of the plot, moving anti-clockwise. In the blue boxes, the difference in colour-intensity is a function of speed, *i.e.* the higher the speed, the stronger the intensity. The plot demonstrates the vehicle movement after 51^{st} seconds.

4.3.1.1 Motion Results

In Figure 4.13, a plot of the vehicle movement is presented. This figure shows the results, including heading, from the self-driving vehicle using the Structured Road Method where the intensity of the blue blocks represent different velocity. Higher intensity means higher velocities. The most visible velocity differences are at the initial position and in the first curve. In Figure 4.13, the simulations were run in 20 Hz and the vehicle data is extracted over 51 seconds with extractions at every 2 seconds.
Figure 4.14 shows extraction over 59 seconds of simulation, where the pedestrian has reached the road, from above. Clearly, the vehicle speed is reduced. Seen in the simulations, the vehicle stopped for the obstacle. Figure 4.15 is the full simulation time from initial position to the parking area. Once the pedestrian was not detected by the camera, the vehicle started to accelerate. The velocity was slower in the cross-section, seen around $(220, 25)$.

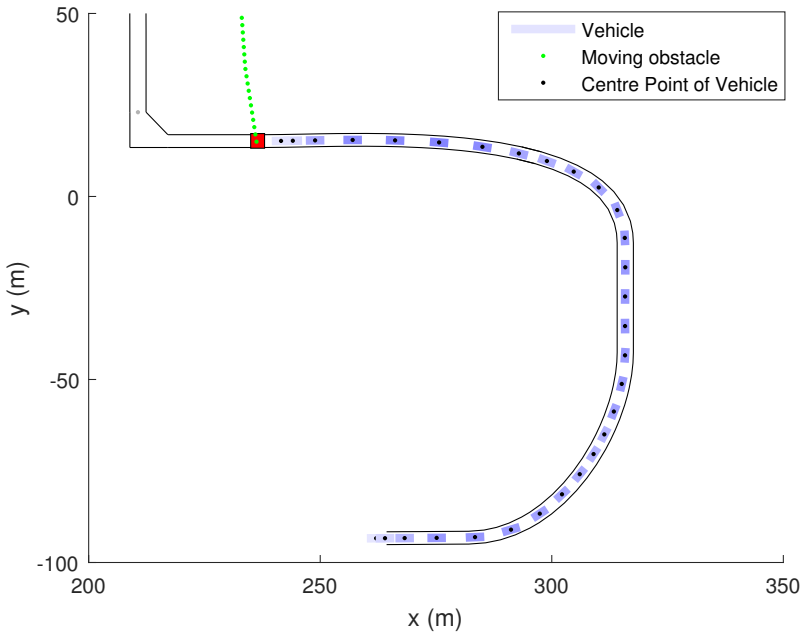

Figure 4.14: Illustration of 59 seconds run-time in simulation. The pedestrian has reached the pedestrian-crossing and the colour-intensity of the vehicle is bright, in front of the crossing. In the last point extraction, the vehicle is reaching stationary state.

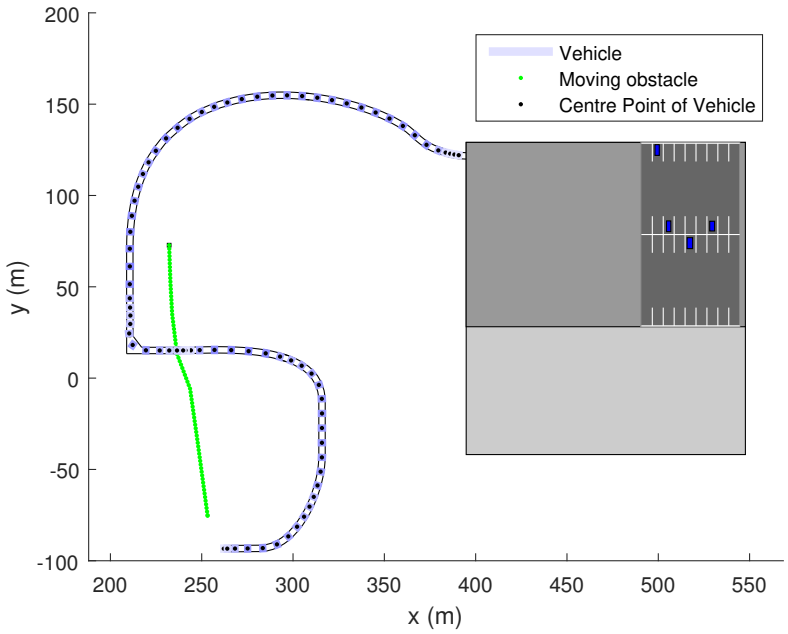

Figure 4.15: A full run-time of the simulation, from start to goal. The colour-intensity of the blue boxes turns very bright in the end, close to the parking area, as the velocity reaches $0ms^{-1}$. The velocity is reduced at the cross-section, at $(225, 20)$.

4.3.1.2 Path Generations

From Figure 4.16, the trajectories are plot with the vehicle, every 4^{th} second for a run time of 76 seconds. The trajectories are plainly fit along the road from the initial positions to the closest station greater than 3 meters from the centre point for every extraction. In Figure 4.17, the data from the simulation is extracted every second. As the motion-controller is outside the main focus in this thesis, it is seen that the motion of the vehicle lacks smoothness in the cross-section, around $(220, 20)$, however, the trajectories are well generated as desired.
Figure 4.18 illustrates the whole simulation time from initial position to the parking area, where the data is extracted every 4^{th} second.

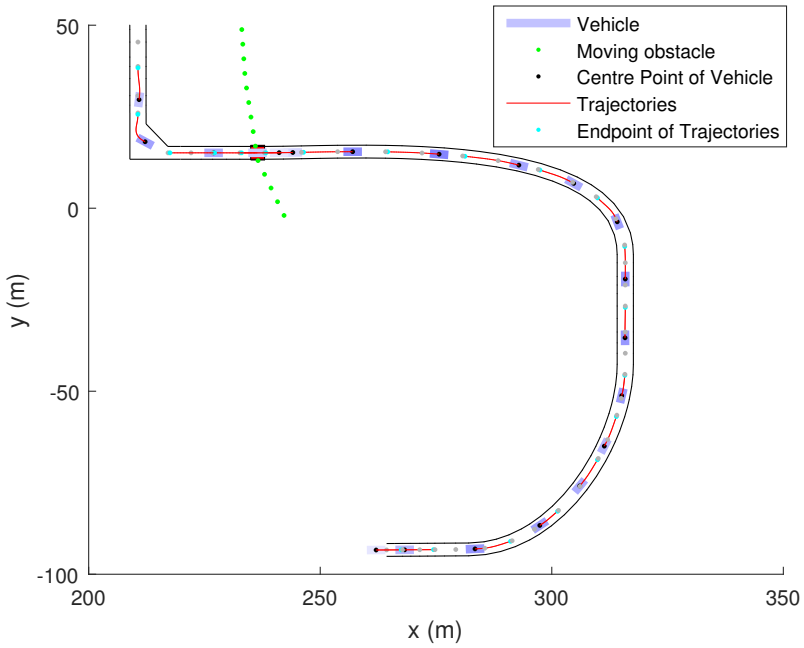

Figure 4.16: Illustration of generated trajectories during simulations. The red lines are the trajectories at different iterations, the cyan points represent the trajectories' end. The grey points are the stations, *i.e.* the given waypoints. Points at every 4 seconds are extracted for a total of 76 seconds run-time of the simulation.

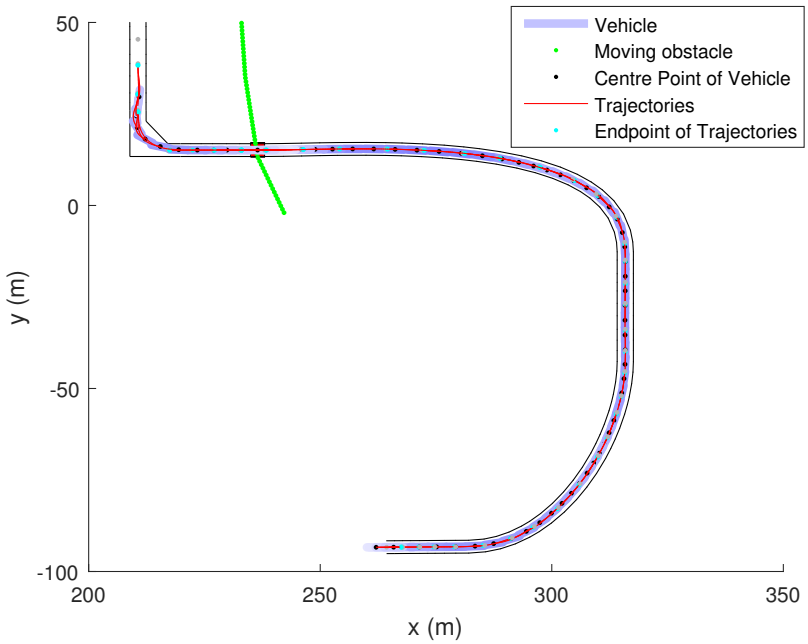

Figure 4.17: Illustration of generated trajectories during simulation of 76 seconds. Now, points are extracted from every second of the simulation.

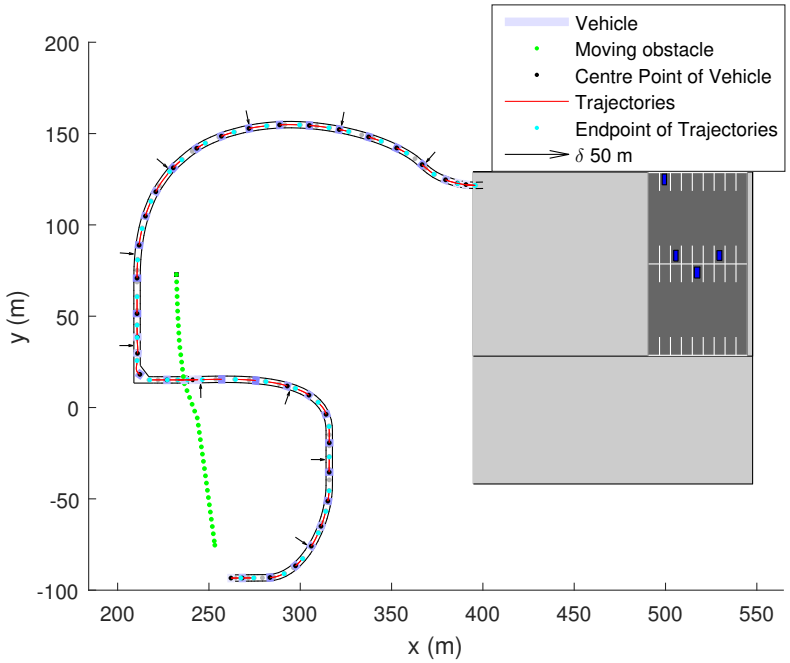

Figure 4.18: Illustration of the generated trajectories, from start to end. Points are extracted every 4^{th} second. The black arrows are indicating every 50 metres of the road.

4.3.1.3 Trajectory Generation

The velocity profiles with respect to the travelled distance is illustrated in Figure 4.19. Concluding the graph, the velocity is reduced in the curves, although, not early enough. This is due to the short horizon. It is seen that the vehicle reaches stationary state for the moving pedestrian and accelerates once the pedestrian has moved out from the detection field. The reference velocity is $6ms^{-1}$, but due to the aggressive curves during the path to the end, the velocity is observed to be lower in some areas.

Figure 4.20 shows a 3-dimensional plot of the trajectories of the first half of the navigation, where the Z-axis represents the velocity. Again, the deceleration before the pedestrian is observable as the smooth acceleration. Examining the straight areas of the road compared to the curved parts, the velocity appears to be higher in the straight road segments. Figure 4.21 shows the second half of the navigation until the vehicle has reached the destination. The lack of a well-tuned controller is noted in these graphs, as the centre points of the vehicle are deviating from the generated trajectories for some extractions of the data. The deviation is most apparent when the obstacle is observed.

Further results are seen in the Appendix A.

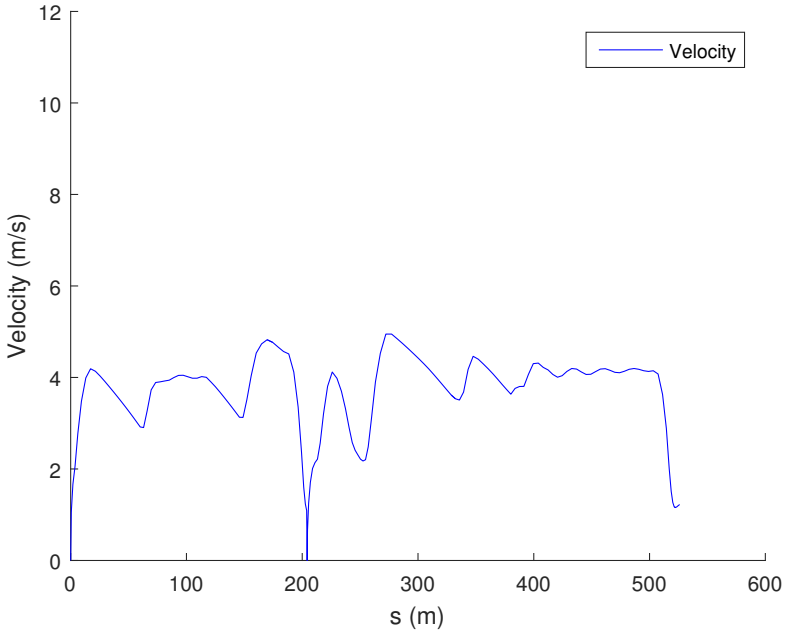

Figure 4.19: Illustration of the velocity behaviour for the whole track. The velocities are extracted every second. Comparing this plot with Figure 4.18, it is seen that the velocities are reduced in the curves, although, not early enough due to the short horizon.

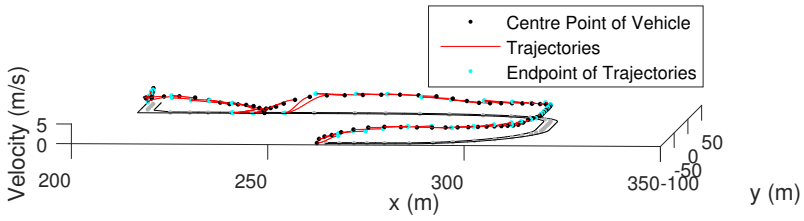

Figure 4.20: Illustration of the generated trajectories including velocities. The trajectories are extracted from initial position to the first half of the route, *i.e.* from $(260, -90)$ to $(210, 30)$ in Figure 4.18. The samples are extracted from every second. The black points represent the location of the vehicle, the blue points represent the end of each trajectory and the red lines are the trajectories. The initial velocity is $0ms^{-1}$ and reaches up to $5ms^{-1}$ at maximum.

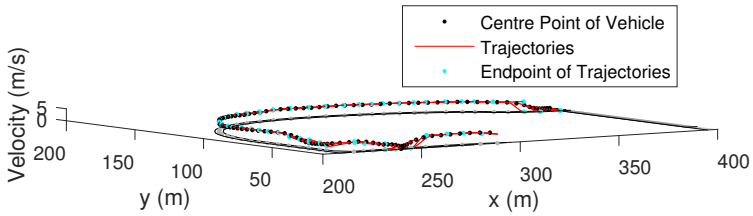

Figure 4.21: Illustration of the generated trajectories including velocities. The trajectories are extracted from 2^{nd} half to end, *i.e.* from $(285, 14)$ to $(400, 120)$ in Figure 4.18. The samples are extracted from every second. The vehicle brakes before the moving obstacle.

4.3.2 Online Unstructured Road Method

For the unstructured environment, the inputs set are showed in Table 4.5. From the generated paths, it is observed that they are smooth for the vehicle to follow. The obstacles are avoided in the offline generation of the path points. In case of reversing, the A* path is generated to the sub-point and then to the goal. The path points are extracted from the total path. By this approach, it is possible to avoid any obstacles around the sub-point. In case of forwarding, the A* path is directly used to extract path points from it. These points are followed by the vehicle by generating curves between the points. Based on the initial and final angle available in the path points, the curves are generated for forward or reverse curves. It is necessary to change the gear when the vehicle has reached the point after which it should reverse. This is solved by setting the gear to be +1 (which means that the forward gears are active) if the next path point is in front of the vehicle and -1 (for the reverse gear) if the next path point is behind the current position of the vehicle. From the test cases it is observed that the obstacle coordinates are to be well-defined. Since the map is not generated as in the traditional A* path planning, it is necessary to have all the coordinates defined accurately. For simulation, a constant velocity is set to $3ms^{-1}$. An observation made is that, in the simulation environment the wheel angle and velocity are opposite for forward and reverse motion. Therefore, while reversing it is necessary to set conditions such that negative values are fed to the controller.

Table 4.5: Parameters for Simulation Test.

Parameter	*Value*	*Unit*
Initial Position	(400,122)	*m*
Final Position	(529,125)	*m*
Initial Angle	0	*degrees*
Final Angle	270	*degrees*
Length of the Vehicle	4	*m*
Safety Distance	2	*m*

A path is generated form the current position at every instance of the simulation. It relies on the controller to make the vehicle follow the generated path. In Figure 4.22, the paths generated at every 4^{th} second are seen. The path followed by the vehicle is seen to be smooth for the vehicle to follow. The paths during the reversing action can be seen. The acceleration, while reversing, is set to 0 to avoid higher curvature of the path. Thus at constant velocity it reverses and decelerates once it is close enough to the desired position.
Clearly, the vehicle does not necessarily go to every next path point, as seen in Figure 4.23. This is due to the condition that the vehicle chooses the next path point if the immediate next path point is too close for the vehicle to generate the Reeds-Shepp curve. This is because, if the Reeds-Shepp curve is to be generated

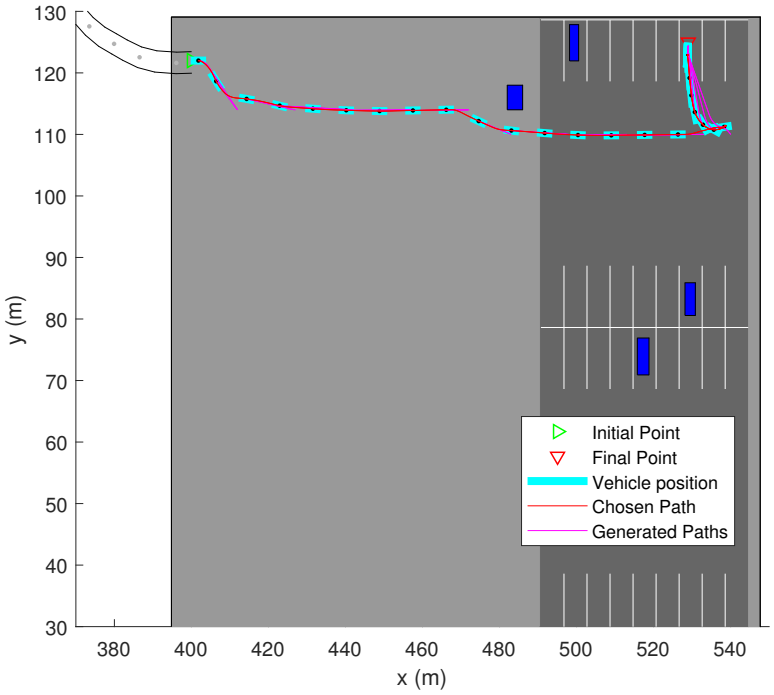

Figure 4.22: Illustration of the trajectories generated in the simulation environment. The trajectories generated in the simulation are at every 4^{th} second, from each path point. The green and red markers represent the initial and final point of the vehicle. The initial position is $(400, 122)$ with an angle of $0°$ and the final position is $(529, 125)$ with a desired angle of $270°$. The blue boxes are static obstacles present in the parking environment.

until it reaches the path point, it might lead to generation of complicated curves. Therefore, choosing the next path point will avoid this and hence maintain the flow for the path.
In Figure 4.24, a more graphic representation of the chosen path is depicted. The path that the vehicle follows from its current position at every 4^{th} second in the simulation, is seen. The simulation runs at a frequency of $20Hz$.
The velocity profile, seen in Figure 4.25, depicts the velocity of the vehicle at every point along the path. In the end of the figure it is observed that the velocity goes to negative values. This is because the simulation environment requires positive values to be fed to the controller for the vehicle to move in forward direction and negative values for the opposite (reverse) direction.
Since it is sufficient for the vehicle to move in a constant speed, a threshold can be set such that the vehicle decelerates if the value of the threshold velocity is exceeded. A threshold of $\pm 3ms^{-1}$ is set so that the vehicle also moves in reverse direction.

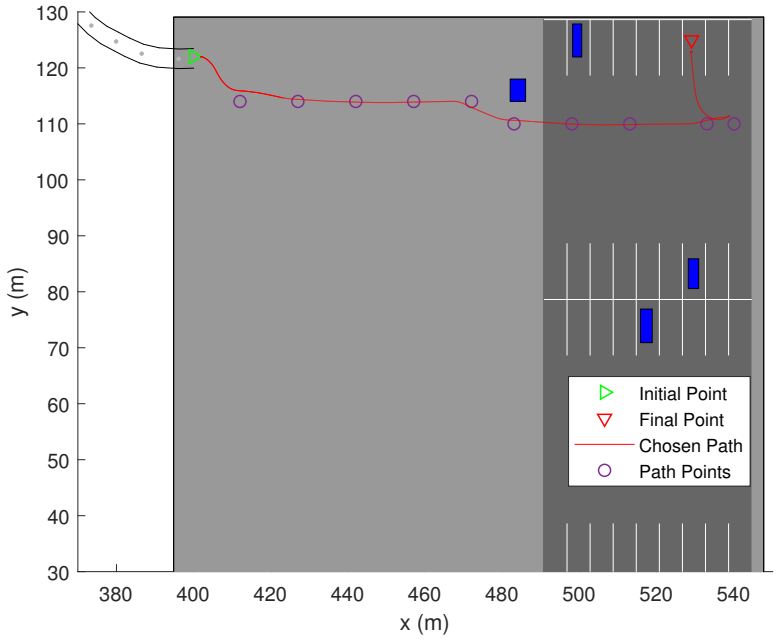

Figure 4.23: Illustration of the path taken by the vehicle to the goal. It is seen that the path is followed based on the trajectories generated from the algorithm. By setting an saturation limit to the wheel angle, the vehicle can be avoided from taking sharp turns. In this depiction, it successfully avoids the crossing of the priority lines and safely enters into the desired position. The offline path obtained is first generated to the sub-point and then to the goal. In this way, any object present around the sub-point is also avoided. The purple circled markers indicate the path points obtained offline.

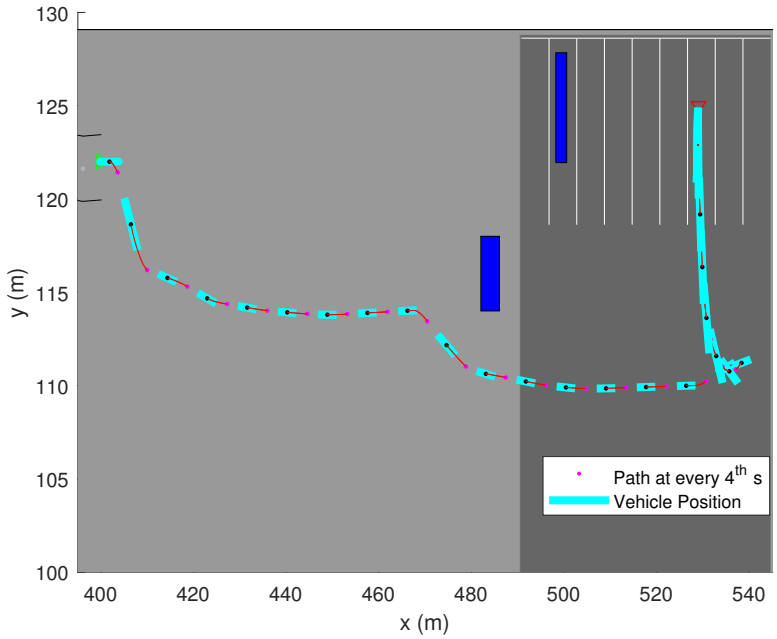

Figure 4.24: Illustration of the vehicle position and its path for the next 40 metres from that position until it reaches the goal. The magenta markers denote the position on the chosen path at every 4^{th} second.

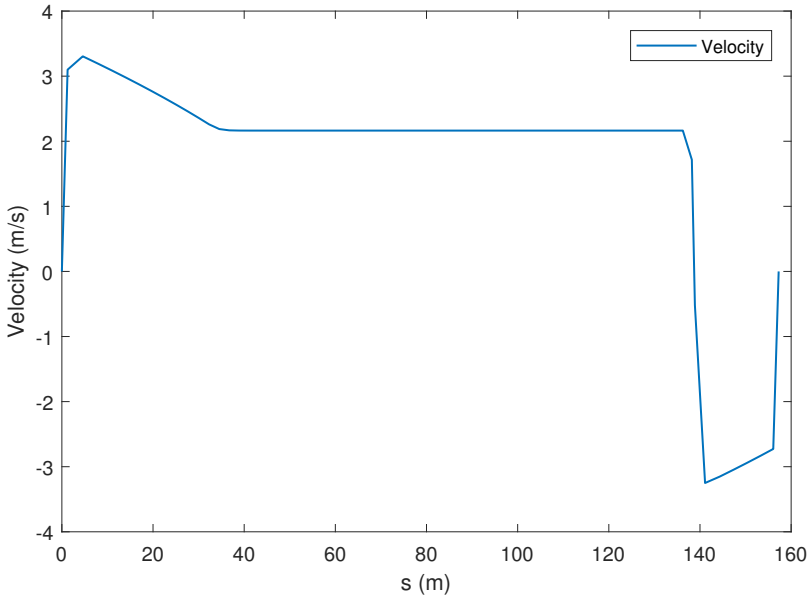

Figure 4.25: Illustration of the velocity of the vehicle along the path. It is observed that the negative velocity refers to the reverse motion of the vehicle in the simulation environment. The overall depiction of the velocity is with respect to the arc length along the path. The threshold on velocity set for this case is $\pm 3ms^{-1}$.

5
Discussion

In this chapter, the results from Chapter 4 are discussed. The aim is to evaluate the algorithms and present a better understanding on the implementations. The different results obtained in Structured Environment Method (offline and online) will be mentioned in Section 5.1. In Section 5.2, the views on Unstructured Environment Method will be discussed. The possible alternatives for the drawback will be mentioned in the respective Sections.

5.1 Structured Environment Method

For the results of the Structured Environment Method, the same cost function is used for the offline generation as well as for the online generation to keep consistency. The only differences are the configurations of horizon, amount of speed profiles and augmented waypoints. The achieved results for both the offline and online generated trajectories have the same behaviour, although the online generated trajectories are generated at every iteration. The offline trajectories are used for cost function tuning and evaluation before the online generation.

5.1.1 Offline

By increasing the amount of augmented vertices, see Equation (3.5a) to Equation (3.5d) and Equation (4.1), it is possible to have a more efficient and smooth planning algorithm. The trade-off lies between accuracy and computational time. However, this is highly dependent on the observed road width. The chosen values in Equation (4.1) are not very practical in the real world and only used for demonstration, as the car could practically move outside the lane boundaries.
Seen in the Equations (4.2), the choice of the P-vector is different to the method. The performance of the algorithm experienced more accurate paths according to the road when the actual values of Equation (4.2) were chosen, especially in Equation (4.2d) that should consider the vertex curvature value. The values of Equations (4.2b) and (4.2c) are supposed to be initial guesses. By setting these values as 0, the path generation succeeded as desired as there were no *a priori* guesses for these values. Regarding the value in Equation (4.2e), this value was chosen as the given waypoints are assumed to be 2 metres apart, along the road. The amount of waypoints were halved due to the tight distance.
One way to generate more consistent velocity profiles throughout the trajectory generation would be to also include different values for a_1 and a_0, seen in Equation

(4.3d) and (4.3e). This is to match the incoming (or current acceleration) and outgoing accelerations. This would require more trajectories and further computational demands.
The cost function weights in Table 4.1 are chosen such that the chosen trajectories satisfy the requirements in generic environments. These can be tuned to match different environments depending on the desired behaviour for the vehicle. In this context, the obstacle weights were given superior values due to the importance of safety. Second highest weight was given to the lateral offset from the centre point to ensure that the vehicle stays as close to the centre of the lane as possible. The weight for centripetal acceleration is high to satisfy smooth navigation of the vehicle (considering the value for centripetal acceleration to be very high in general). A constraint on the acceleration is included to get as smooth accelerations as possible, except in emergency situations, obtained in the online results. This implies that if an obstacle is closer than a certain distance to the vehicle, the deceleration ignores the threshold.

In Figure 4.1 and 4.2, the horizon is 1 station. Seen in this configuration, the steering ratio gets slightly aggressive when avoiding the obstacle. The behaviour when using a horizon of 2 stations is exactly the same, seen in Figure 4.3. By tuning the cost setup more carefully, a desired behaviour would be to revert to the centre of the lane via two stations, after the obstacle, instead of doing so instantly; in Figure 4.3 between station 7 and station 8, where the desired manoeuvre would be from 7 to 9. This occasion is due to too high weight for w_{lat}. The major difference in these scenarios is the velocity profile, see Figure 4.2 and 4.4. The reason that the velocity is not reaching $10ms^{-1}$ is that the cost for C_s is affecting less than C_a, C_j and C_{ca} from Table 3.2, since the acceleration and jerk is zero and the total cost is lower. However, the velocity reaches the reference velocity, $10ms^{-1}$ in Figure 4.4 as 2 stations are considered and the total cost is cheaper of choosing the longer trajectory segment. The critical distance choice, $t_{distance}$ is set to 1.5 meters for demonstrational purposes.
In Figure 4.6, the trajectory choices are as desired. An obstacle is placed in the middle of the road where the vehicle would hit the obstacle in any vertex choice. The only way to avoid the obstacle is to decrease the speed to $0ms^{-1}$ before the obstacle. The trajectories are accelerating up to reference velocity posterior to the obstacle avoidance. Depending on the scenario, it can be decided to keep the velocity at 0 posterior to the obstacle.
Until now, horizons are a critical choice of the future vehicle behaviour. Characteristics can be decided by either increasing the waypoint distance or by computing dynamic programming to more than 2 stations (increasing horizon). As seen in Figure 4.7, too long distance between the waypoints results in road boundary crossing, although the velocities would be better adapted to the curved roads.

5.1.2 Online

One difference between the offline generated results and the online generated results is the lack of sensor in the offline part. In the offline section, only stationary obstacles were considered whereas in the online section, a dynamic obstacle was included.

There are different ways of detecting objects for an autonomous system, yet in this part a obstacle detection camera was used to measure the Doppler velocity of any object in front of the vehicle, which is not possible in the offline section.

In Figure 4.14, the motion of the vehicle stays within the boundaries of the road and brakes for a detected obstacle. The problem with the current solution is that if the camera would detect a stationary obstacle, the vehicle would stop anyway. Although, as this is an issue concerning the Decision Layer of an autonomous system, only a simple case was considered in this thesis, as classification and prediction is outside this scope.
The generated trajectories are well fit to the road curvature in Figure 4.17, yet the vehicle is not tracking the generated trajectories as desired in the cross-section in $(210, 20)$ and needs optimisation. This is due to the simple controller. Looking at the velocities during the simulation, in Figure 4.19, the acceleration is smooth, despite the short horizon the vehicle is able to slow down when the curvature of the road is high. The reference velocity, which is set to $5ms^{-1}$ is reached when the curvature of the road is reduced. Seen in Figure 4.18 and 4.19, the velocity is greatly reduced in the cross-section, at the travelling distance of 250 metres .
Since the trajectory generation is updated at every iteration, small jumps of the desired velocity may occur due to different choices of trajectories in the cost function, at each iteration. One way to overcome this problem is to, slightly and carefully, reduce the last generated trajectory's cost such that the chosen velocity profile at each moment keeps being consistent.
Another con regarding this algorithm is the critical waypoint placement. As no other parameters are considered in this type of algorithm, the algorithm is dependent on well-placed waypoints.
When testing this algorithm in other experiments including other vehicles in front of the ego-vehicle, the ego-vehicle manages to stay behind the leading car without hitting it. This is mainly due to the Detection Camera and the measured Doppler velocity, where the leading car's velocity is considered to be less than the reference velocity.

5.2 Unstructured Environment Method

The paths to be followed in an unstructured environment can be obtained by method explained in Section 3.2. There are alterations that can be performed to make it even more generic. As seen in Figure 4.10, the sub-point has to be generated during the offline A* path calculation. This can be made generic by setting conditions for all possible heading angles that a vehicle can have when it reaches a distance to generate a sub-point. Thus, by generating a consecutive offline path to a sub-point and then to the goal can produce generic results. But this might be redundant in cases where just forward parking is sufficient, such as in Figure 4.8. The conditions for generation of sub-point can be chosen to be in offline-calculation, A* path points extraction, or online-calculation, Reeds-Shepp curve generation. The advantage with having it offline is that the obstacle avoidance can be done in the former method, whereas in the latter approach the algorithm works in a more generic way but does not

guarantee safety around place where the the sub-point lies.

The information about existence of obstacles is to be provided by the perception sector, as *a priori*. The most important thing for offline computation is this information, which should also include information regarding the priority lines and the parking spots, where the vehicle should not traverse. This is because, depicted in Figure 4.22, the Reeds-Shepp curves does not obey the constraints of obstacle avoidance. Thus, it is a safety requirement to have the path points placed as far from the obstacles as possible. In a practical environment with dynamic objects, the condition set to avoid such obstacles is by reducing the velocity through the perception from sensor data. But, as it reaches a distance that is closer than the safety distance, the vehicle stops. In this case replanning has to be done offline and the new path points are to be fed to the vehicle to follow.

Performing manoeuvres in tight spaces, such as full parking lots or similar environment, forces the algorithm to generate curves assuming the environment to be structured. Thus, in this case it is necessary to acquire path points that consist information about the heading angles of the vehicle at its current position and its final position at every path point. In such a complex case, it is necessary to consider the vehicle's turning radius, obstacle coordinates, angle differences at every point. A case in which the safety distance to the obstacles, during offline calculation, is mandatory can be seen in Figure 4.9. Considering the path obtained in this Figure, the vehicle might barge into obstacles if the safety distance is not sufficiently maintained. A more efficient way would be to follow the algorithm used for Structured Road Method, mentioned in Section 3.1, which generates trajectories to the next waypoint. These trajectories are also safe to follow since they are affected by cost functions which weigh every different parameter that is important for that particular manoeuvre.

In Figure 4.24, it is seen that the vehicle takes the shortest possible path, which is to go towards the obstacle and then avoid it once it is at a vicinity of the safety distance. Although, it does not seem to be as in a practical case, in which a person would not go towards the obstacle. The desired goal position is to be given to the algorithm while a human being would look around for a free spot and park the vehicle in another spot. Thus, this algorithm can be adapted to also search for a possible and valid goal position based on the sensor data obtained.
An optimal path generation algorithm can be achieved if it is possible to implement obstacle avoidance in Reeds-Shepp curve generation. By this method, the offline calculations can be completely avoided, although the time to reach the destination might be more than the optimal time. Eventually, the cost functions of this upgraded algorithm can comprise of constraints on the curvature, dynamic obstacles, static obstacles, jerk and other factors.

5.3 Merging Algorithms

As the Reeds-Shepp's curves and the cubic polynomials generated in this thesis, serve the same purpose, an interesting experiment would be to use Reeds-Shepp's curves in the Structured Road Method and cubic polynomials in the Unstructed Road

Method. One constraint would be the issue of generating reverse paths using the cubic polynomials. As the Reeds-Shepp's algorithm does this automatically, doing so with cubic polynomials would be more complex. Reeds-Shepp's or specifically Dubin's paths were not tested for the Structured Road Environment. The problem of using Reeds-Shepp's or Dubins' paths in structured environment is that the path do not always have to be the shortest, the algorithm needs more optimisation and small modifications.

6
Conclusion

In this Chapter, the focus of the results in the overall autonomous classification is mentioned. The advancements in the implemented algorithms is discussed.

The reason these two algorithms were inspired from State Lattice and Hybrid-A* is due to the potential that was seen in the algorithms during the short literature study. The final results achieved could have been more accurate with less computational demands using other algorithms. Another area which showcases well performed results is the use of traditional Model Predictive Control, as this seems to be the state-of-the-art approach today. The State Lattice was chosen due the advantage of computational time. In this thesis, optimisation of the code was not within the scope, therefore, the computational time of both algorithms are more than necessary. Without any decision making layer, only simple scenarios can be created since the object classification and prediction part is excluded along with other rules of the road than the presented ones. The architecture of autonomous vehicle systems varies depending on the different developers. The trajectory generation layer is a critical part of an autonomous system and due to continuous environmental changes, this layer is usually run in a high frequency, hence the computational time of the trajectory generation algorithm is required to be not more than a few milliseconds. The Structured Road Method algorithm is dependent on the amount of augmented vertices, amount of velocity profiles and choice of horizon, which results in a diverse computational time relying on the configuration. Conclusively, the overall trade-off lies between having an accurate trajectory generation and having high computational demands. The real-time results of the Structured Road Method used in Section 4.3 ware set to use a horizon of 1 station and no augmented vertices, together with 14 velocity profiles. The experiments were conducted on a 64-bit Operating System consisting of an i7 processor with 8GB dedicated graphics-card and a RAM of 16GB. This setup resulted in a trajectory generation less than 0.5 seconds, although, when the horizon and augmented vertices were increased, the computational time became longer. As code optimisation was excluded in this thesis, this is one of the major drawback regarding the developed algorithm.
The Structured Road Method was seen to work generically. Given waypoints along any road, the algorithm could perform trajectory generation where the trajectories were well fit to the roads and the chosen speed profile showcased smooth accelerations.

In the Unstructured Environment method, the main aim was to implement path generation such that the trajectories avoided static and dynamic obstacles and parked the vehicle by reversing. The paths generated according to the method explained

in Section 3.2, can avoid the static obstacles and also enable reverse parking but avoiding dynamic obstacle might be a critical case, in which the algorithm might time-out if the dynamic obstacle is in front of the vehicle for a longer period of time. The approach of this algorithm does not provide generic trajectory generation. This is mainly due to the dependency on path points obtained from the chosen path search method. The Reeds-Shepp curves do not avoid obstacles, which is also another reason that leads to re-planning of the path. Considering complex environments, it is necessary to validate the path points that are obtained during the offline computation. There is no fixed method to generate the sub-point before reversing, which in practical cases is decided based on the space available in the vicinity of the parking spot.

The motion controller in this thesis was developed simply to visualise practical results. The algorithms were tuned together with the current controllers. A better controller is needed to demonstrate the algorithms' true capacity and a consequence of this is that the vehicle had difficulties to follow trajectories, in *e.g.* cross-sections, where sharp turns were desired. Seen from the results, the presented outcome seemed to be promising. Although by investing more time in the tuning process of the parameters of the algorithm, the navigation of the ego-vehicle could satisfy the smoothness, efficiency and safety criteria better.

The ultimate aim of this thesis has been to attain an autonomous valet parking functionality. This has been experimented by combining the algorithms such that the Structured Environment Method navigates the vehicle to the parking lot or the depot and further, the Unstructured Environment Method parks the vehicle in the assigned spot. In simulation, this combination works well such that the final point of the navigation path is set to be the initial point of the parking path. Although, path generation in the unstructured environment needs more improvements in terms of computation and testing in order to be generic in cases such as dynamic environments.

Since this algorithm allows generation of reverse paths, it is also possible to apply this approach for reverse parking of busses or trucks attached with trailers. Although, the dynamic model of these systems would be a challenge to formulate before planning a suitable path. The algorithms works well for trucks, as for cars. The difference between these vehicles is the dynamics, which these algorithm do not consider when generating trajectories. The only parameter taken into consideration is the wheelbase of the vehicle.

6.1 Future Work

For future work in this thesis, a few possible suggestions for advancements and features that can be added to enhance the current results are discussed. It is seen that there are various ways to enhance the autonomous behaviour.

Addition of lane marker sensors, that detect and provide information about the lanes, for the algorithm to generate the cubic polynomials can enable automatic generation of waypoints as the vehicle moves along a structured lane road. Additional benefits

of using lane marker sensors is correction of waypoint placements and lane centre estimation since in reality, the other sensors will introduce noise and experience disturbances. This will make the algorithm more efficient and accurate even if the given waypoints are deviated from an ideal approach in terms of lane detection. Another approach for the same method that can be included is implementation of quartic or quintic polynomials. As the use of quartic or quintic polynomials allows further kinematic conditions, such as curvature rate and its derivative that may result in better continuity of these conditions, as well as smoother paths. This would require more computational time. Further, fine tuning of the existing parameters in this algorithm can enable an even smoother, more efficient and safer performance.
Another interesting experiment to consider is the increase of augmented waypoints along the horizon in real time. By obtained coordinates for static obstacles, these can be avoided based on the *a priori* coordinate information obtained from the perception without using any sensors in the online scenario. The complexity of using more waypoints is that the cost function tuning will become more critical as there would be more options to choose between.

The parking algorithm can be upgraded by replacing the A* algorithm with an RRT* algorithm. The RRT* algorithm provides a non-holonomic path which can result in generation of path points that are practically feasible for the Reeds-Shepp curve generation. The steps of the method: generation of A* path, extraction of path points from the A* algorithm and then generation of Reeds-Shepp, can be automated such that feeding the information yields the result of path. This would avoid the offline and online computation. Although, replacing the whole approach by only attaining paths using versions of RRT* algorithm might lead to jagged paths, which are non-feasible.

To further validate the results more scenarios can be tested with these algorithms. The performance of the vehicle can be observed for the different scenarios and hence different parameters can be evaluated. The most critical part of the algorithms would be to optimise the time. The time taken to evaluate the information and provide the paths should, at the most, be $10^{-3}s$ for online path generating scenarios.

The State Lattice method is a potential candidate for future development, thus optimisation in terms of computational demands is needed. This method is highly dependent on the waypoints that are received such that they do not deviate from the lane's centre. The possibility with this algorithm is the generic approach. By augmenting more waypoints, lane changes and car over-taking can be considered. Another optimisation would be to also post-process the final trajectories between the waypoints, *for example*, to minimise curvature and additionally optimise the acceleration.
The Hybrid-A* method showed good results, thus until now the algorithm is rather scenario dependent. As the Reeds-Shepp curve generation is sensitive to short distance path generation, it automatically includes reverse paths in those type of path generations, making this method unreliable. There are other manipulated versions of the Reeds-Shepp method that would be more reliable and fit better in these operations.

Bibliography

[1] Katrakazas, C., Quddus, M., Chen, W-H., Deka, L. (2015). Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation research* ***Part C***, 60, 416-442.

[2] Paden, B., Čáp, M., Yong, S-Z., Yershov, D., Frazzoli, E. (2016). A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1), 33-55.

[3] Leonard, J., How, J., Teller, S., Berger, M., Campbell, S., Fiore, G., Fletcher, L., Frazzoli, E., Huang, A., Karaman, S., et al. (2008). A Perception-Driven Autonomous Urban Vehicle. *Journal of Field Robotics*, 25, 1-48.

[4] Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M., Dolan, J., Duggins, D., Galatali, T., Geyer, C., et al. (2008). Autonomous Driving in Urban Environments: Boss and the Urban Challenge. *Journal of Field Robotics*, 25, 425-466.

[5] Pivtoraiko, M., Knepper, R., Kelly, A. (2009). Differentially Constrained Mobile Robot Motion Planning in State Lattices. *Journal of Field Robotics*, 26(3), 308-333.

[6] McNaughton, M., Urmson, C., Dolan, J., Lee, J-W. (2011). Motion Planning for Autonomous Driving with a Conformal Spatiotemporal Lattice. *IEEE International Conference on Robotics and Automation*.

[7] McNaughton, M. (2011). *Parallel Algorithms for Real-time Motion Planning* (Doctoral dissertation, Carnegie Mellon University). Retrieved from https://www.semanticscholar.org/.

[8] Xu, W., Wei, J., Dolan, J., Zhao, H., Zha, H. (2012). A Real-Time Motion Planner with Trajectory Optimization for Autonomous Vehicles. *IEEE International Conference on Robotics and Automation*.

[9] Howard, T (2009). *ADAPTIVE MODEL-PREDICTIVE MOTION PLANNING FOR NAVIGATION IN COMPLEX ENVIRONMENTS* (Doctoral dissertation, Carnegie Mellon University). Retrieved from `https://www.semanticscholar.org/`.

[10] European Commission. 2017. *Statistics – accidents data*. [ONLINE] Available at: `https://ec.europa.eu/transport/road_safety/specialist/statistics_en#`. [Accessed 29 June 2018].

[11] National Highway Traffic Safety Administration. (2017). *2016 Fatal Motor Vehicle.* [ONLINE] Available at: `https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812456` [Accessed 29 Jun. 2018].

[12] National Highway Traffic Safety Administration. (2015). *Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey.*

[ONLINE] Available at: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115 [Accessed 29 Jun. 2018].

[13] Shoup, D. (2007). Gone Parkin'. *The New York Times.* [ONLINE] p.A25. Available at: https://www.nytimes.com/2007/03/29/opinion/29shoup.html [Accessed 29 Jun. 2018]

[14] Gu, T., Dolan, J.M., Lee, J-W. On-Road Trajectory Planning for General Autonomous Driving with Enhanced Tunability *Carnegie Mellon University, General Motors*

[15] Ferguson, D., Howard, T.M., Likhachev, M. Motion Planning In Urban Environments: Part II *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France (2008).

[16] Dolgov, D., Thrun, S., Montemerlo, M., Diebel. J. Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments, *The International Journal of Robotics Research*, Vol 25, Issue 5, pp 485-501 (2010).

[17] Dreyfus, S.E. An Appraisal of Some Shortest-Path Algorithms, *University of California*, Berkeley, California (1968).

[18] Moses Sathyaraj B., Jain L. C., Finn A., Drake S. Multiple UAVs path planning algorithms: a comparative study, *Knowledge Based Intelligent Engineering System Center and Defence Science Technology Organisation*, Australia (2008).

[19] Likhachev M., Ferguson D., Gordon G., Stentz A., Thrun S. Anytime Dynamic A*: An Anytime, Replanning Algorithm, *Carnegie Mellon University and Stanford University*, USA (2005).

[20] Chrpa, L., Osborne, H. (2014). Towards a Trajectory Planning Concept: Augmenting Path Planning Methods by Considering Speed Limit Constraints. *Journal of Intelligent & Robotic Systems*, 75(2),243–270.

[21] Petereit J., Emter T., Frey C.W., Kopfstedt T., Beutel A. Application of Hybrid A* to an Autonomous Mobile Robot for Path Planning in Unstructured Outdoor Environments, *Fraunhofer Institute of Optronics, System Technologies Image Exploitation and Diehl BGT Defence GmbH Co.*, Germany (2012)

[22] Karaman S., Frazzoli E. Incremental Sampling-based Algorithms for Optimal Motion Planning, *Robotics: Science and Systems*, Spain (2010).

[23] Qureshi A. H., Ayaz Y. Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments, *Robotics and Intelligent Systems Engineering(RISE) Lab and Department of Robotics and Artificial Intelligence*, Pakistan and *Department of System Innovation*, Japan (2017).

[24] LaVelle S. M., Kuffner J. J. Jr. Rapidly-Exploring Random Trees: Progress and Prospects, *Iowa State University*, USA and *University of Tokyo*, Japan (2001).

[25] Naderi K., Rajamäki J., Hämäläinen P. RT-RRT*: A Real-Time Path Planning Algorithm Based On RRT*, *Aalto University*, France (2015).

[26] Palmieri L., Koenig S., Arras K. O. RRT-based nonholonomic motion planning using any-angle path biasing, *IEEE International Conference on Robotics and Automation (ICRA)*, Sweden (2016).

[27] Dubins L. E. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents, *American Journal of Mathematics, Vol 79, No. 3, pp 497-516* (1957).

[28] Shkel A. M., Lumelsky V. Clossification of the Dubins set, *Robotics and Autonomous Systems 34, 179-202*, USA (2001).

[29] Chitsaz H., and LaValle S. M. Time-Optimal Paths for a Dubins Airplane, *46th IEEE Conference on Decision and Control* (2007).

[30] Choi H. Time-Optimal Paths for a Dubins Car and Dubins Airplane with a Unidirectional Turning Constraint, *Aerospace Engineering, The University of Michigan* (2014).

[31] Reeds J. A., Shepp L. A. Optimal paths for a car that goes both forwards and backwards, *Pacific J. Math. 145, no. 2, 367–393* (1990).

[32] Bicchi A., Casalino G., Santilli. Planning Shortest Bounded-Curvature Paths for a Class of Nonholonomic Vehicles among Obstacles, *University of Pisa* (2010).

[33] Tsai Y-J., Lee C-S., Lin C-L., Huang C-H. Development of Flight Path Planning for Multirotoe Aerial Vehicles, *Aeronautical System Research Division, , Chung Shan Institute of Science and Technology and Department of Electrical Engineering, Chung Hsing University*, Taiwan (2015).

[34] Kolski S., Ferguson D., Bellino M., Siegwart R. Autonomous Driving in Structured and Unstructured Environments, *Scientific Figure on ResearchGate*, Available from:https://www.researchgate.net/Results-from-global-planning-and-mapping-in an-unstructured-environment, (2018)

A

Additional Plots

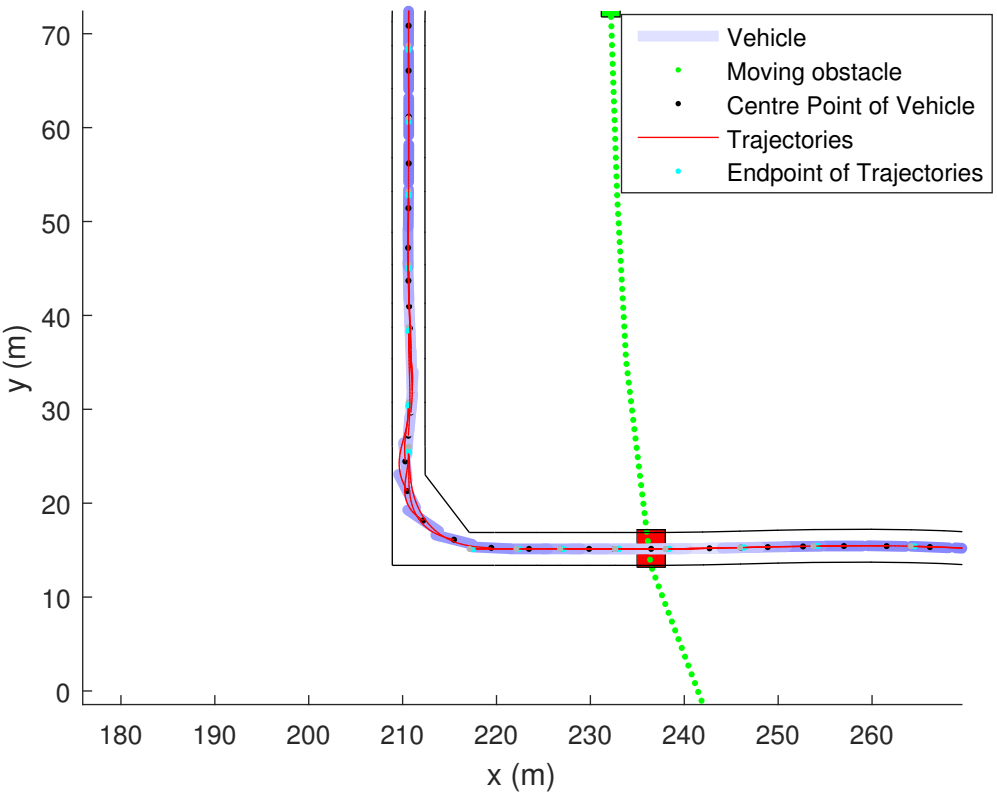

Figure A.1: Illustration of the generated trajectories in a difficult case, the corner. Point extraction frequency of 1 second.

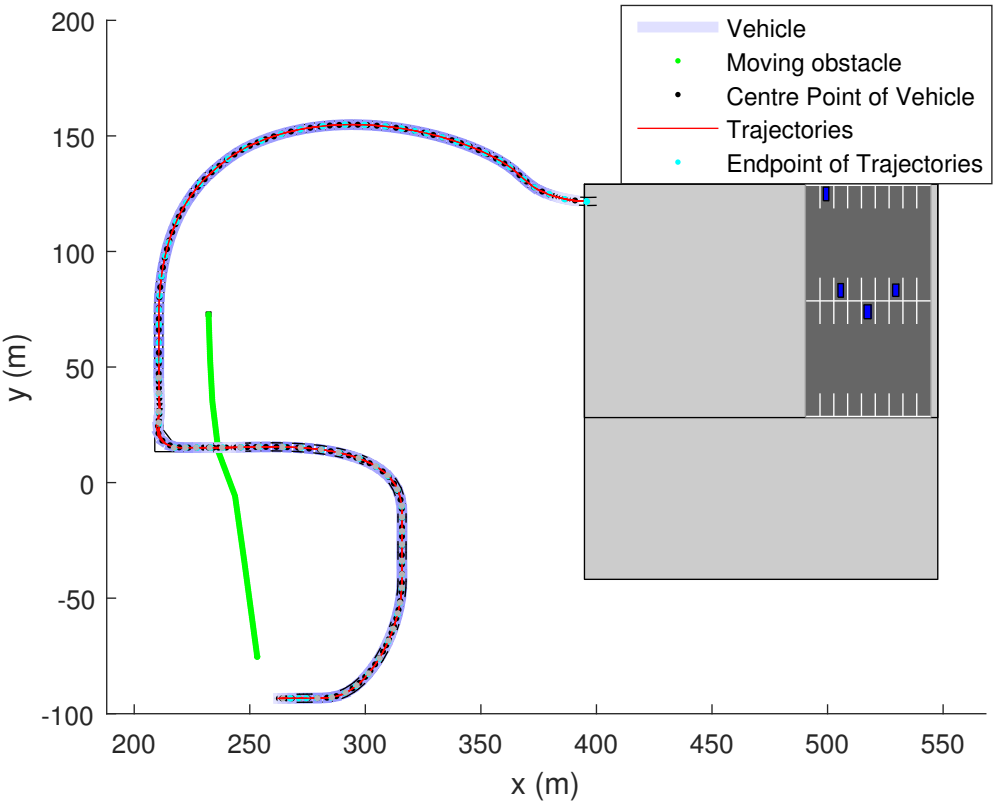

Figure A.2: Illustration of the generated trajectories from start to end. Points are extracted every 1 second.

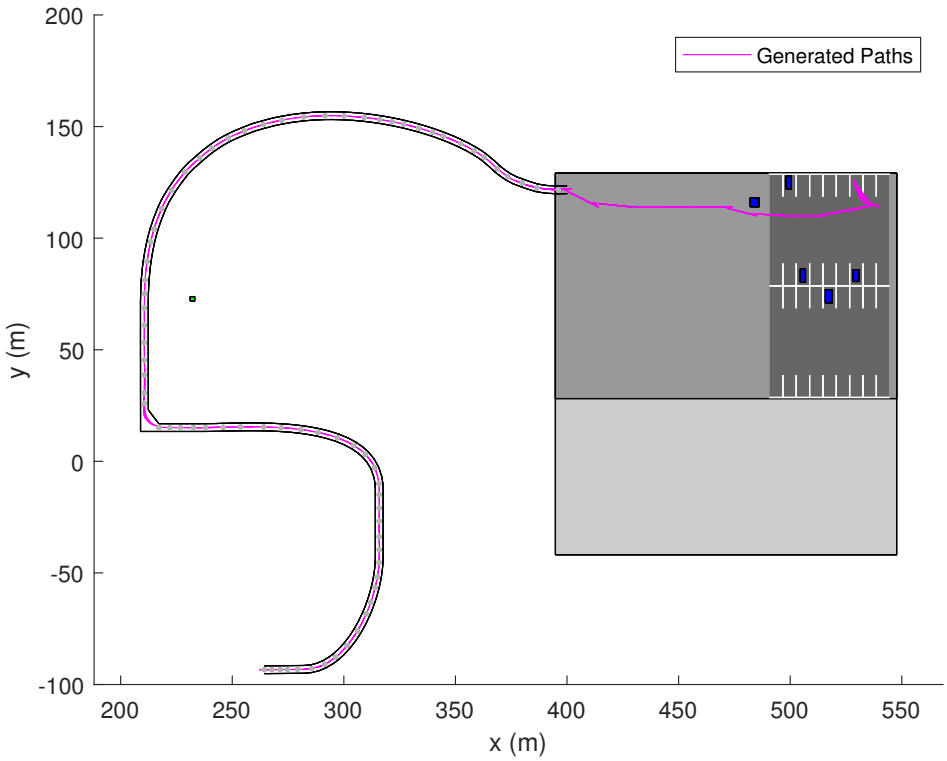

Figure A.3: Illustration of paths generated at a time interval of 2 seconds in the PreScan simulation environment.

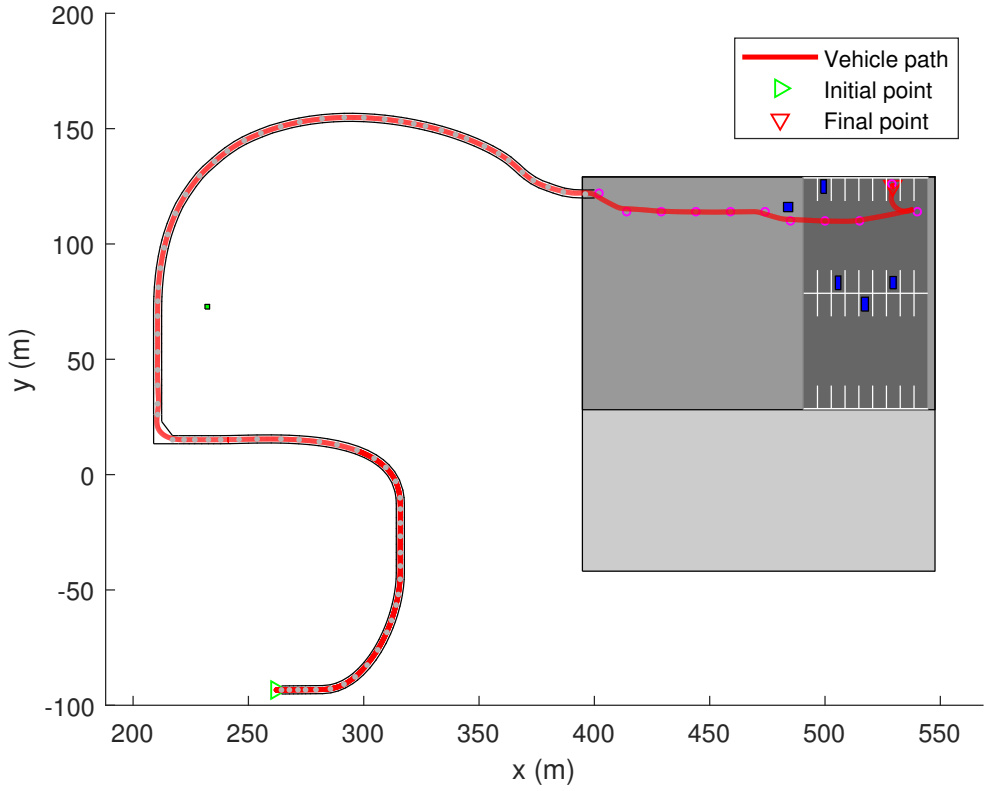

Figure A.4: Illustration of the path obtained in the simulation environment. The initial point of the vehicle is at (262,-93) and the final point is at (529,125) on the plot. Both the algorithms are merged and a smooth transition is seen during the switch of the algorithms. The waypoints for the structured road are highlighted by the white dots and the path points for the unstructured environment are highlighted by magenta circles. The change of algorithm takes place at (398,122) on the plot.

B

Common Algorithms

B.1 Path Finding

A short explanation of a few variants of the A* algorithm can be seen as follows:

- The Anytime D* algorithm, [19], finds a path by not only avoiding known obstacles but also by replanning the path in case there are any unknown obstacles or changes in the map. This follows a more generic approach, making the search to be dynamic in a simulation environment. A downside of this is that, if the constraints for replanning are not well-defined, the path to the goal might be longer or deviated from the map constraints. This methods makes use of the (x, y) position coordinates on the 2-D map to navigate.
- The Augmented A* algorithm, seen in [20], is an A*-like algorithm that includes the dynamics and kinematics of a point-mass-model. This enables generation of feasible trajectories during each search. One major problem in this is its computational heaviness.
- The Hybrid-A* algorithm is an advanced version of A* and gives assurance that the obtained path is the shortest. This also uses two different costs to navigate between the nodes. The continuous search enables a faster search relative to other variants. The major difference in this method is that it considers the continuous nature of the environment making it more realistic to the real environment, [21]. The method makes use of the (x, y) position coordinates and the angle, θ, between two nodes.

Another slightly different approach of the incremental search is the RRT algorithm, found in [22]. In this approach, the map is not evenly distributed into a discretised grid, but is explored based on a random configuration. A stepsize is decided based on the resolution of the map. At every step a sample node is generated in random. A node is selected in the direction of the random node at a distance of the stepsize. The cost is calculated from the near node to the goal node. Clearly, the cost will be higher if the near node is farther or deviated from the direction towards the goal. Similarly, the nodes are generated based on the number of samples and a path is obtained to the goal. It is seen that this method takes more computational time and is dependent on number of samples, also known as *probabilistic complete*. Few advancements of this method are the Rapidly-Exploring Random Graphs (RRG or RRT*), bidirectional RRT* or RRT*-Smart, [24], Theta*-RRT, Real-time RRT* (RT-RRT*). In general, this method is computationally heavy due to requirement of storing the numerous nodes and also keeping track of the contained information. An example of this can be visualised in Figure B.1.

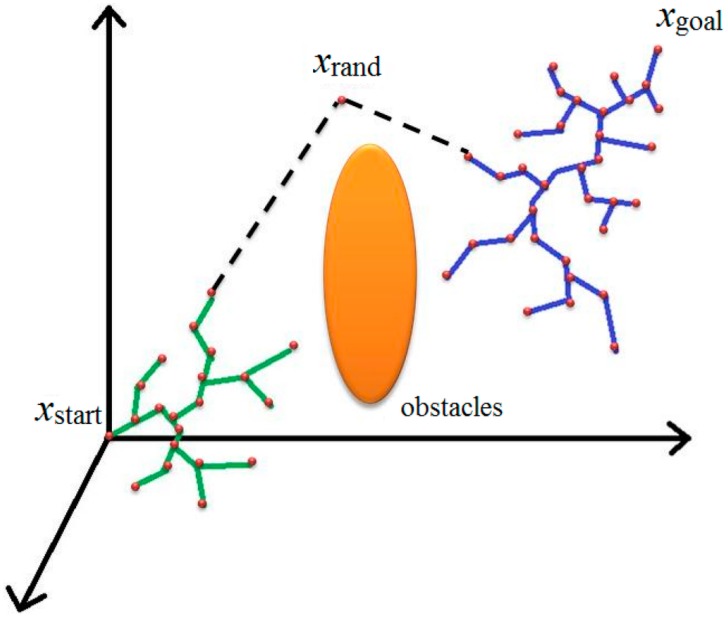

Figure B.1: Illustration of RRT algorithm, showing an example of random node search mentioned as x_{rand} and many near nodes (x_{near}) expanding bidirectionally with a stepsize between x_{start} and x_{goal}, seen as red points. Here, the exploration is seen to be in a three dimensional environment. In this figure, the obstacle is the orange, oval shape in the middle of the map. Image taken from [33].

A short explanation of a few variants of this method can be seen as follows:

- In RRG, [23], the cost of a deviated random node is calculated similar to RRT method, but at the same time the costs of nodes surrounding this node are also set to the same value, therefore avoiding generation of random nodes close to the currently chosen random node.
- In RT-RRT*, [25], the visited nodes are not discarded and hence, this enhances the search to provide an optimal solution in a dynamic real-time environment. Storing the sampled paths for every instance is computationally heavy.
- In Theta*-RRT, [26], a combination of an angle search with generation of a random node is carried out. This method has application in environments with complex non-holonomic constraints.

B.2 Path Generation

Dubins' method, in [27], gives a set of paths consisting of circular arcs and line segments, that can be followed by a vehicle. The proof of this is given by J.A. Reeds and L.A. Shepp in [31]. The set of path also considers the turning angle depending on the vehicle's curvature along each path, as shown in [28]. This method shows that any path can be expressed as a set of segments, called as Dubins' set, which is a sequence *'CCC'* or *'CSC'*, where C and S represent a curve of a circular arc of unit radius or a line segment of a certain length, respectively. There are six admissible

curves, in a Dubins' set. They are

$$\mathcal{D} = \{LSL, RSR, LSR, RSL, LRL, RLR\}, \tag{B.1}$$

where *'L'*, *'R'* and *'S'* represent *'Left'*, *'Right'* and *'Straight'*, [28, 27]. Dubins' paths consider only forward motion of the vehicle. Left and Right mean the counterclockwise and clockwise arcs of radius, equal to the turning radius of the vehicle.

As explained in [30], a brief working of this method is as follows:

- The initial and final coordinates are obtained in the Cartesian coordinate system.
- Using Dubins' scalar equations, found in [27], the parameters that correspond to the length of the segments are obtained.
- This is carried out for each curve until the goal point is reached.
- Finally, the curves are interpolated to obtain a smooth and traversable path.

The Dubins' curves are limited to only forward motion of the vehicle, which can be seen from Figure B.2. These were further extended for generation of reverse curves by J.A. Reeds and L.A. Shepp. Reeds-Shepp's study extended this method to provide solution for a vehicle to be capable of moving forward and backward. The geometric approach of Dubins' path was also improved by Boissonnat, Cerezo, and Leblond using optimal control theory's principle of Pontryagin, which is mentioned in [29]. Balkcom and Mason characterised time-optimal paths to make it capable to move in any direction by considering a unicycle model, while Bakolas and Tsiotras presented an approach to allow different radii for clockwise and counterclockwise turns, as mentioned in [30].

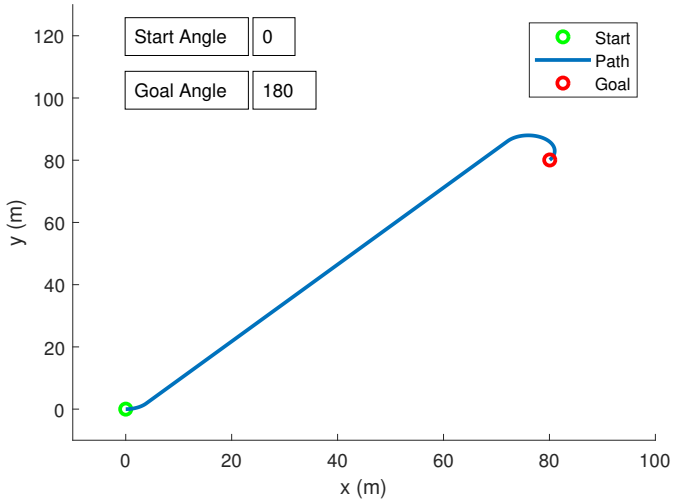

Figure B.2: Illustration of Dubins' Path showing a forward manoeuvre, from the start node to the goal node, given the initial angle (0°) and goal angle (180°). Here, the blue line depicts the path, the initial and final positions are represented by the red and the green circle, respectively.