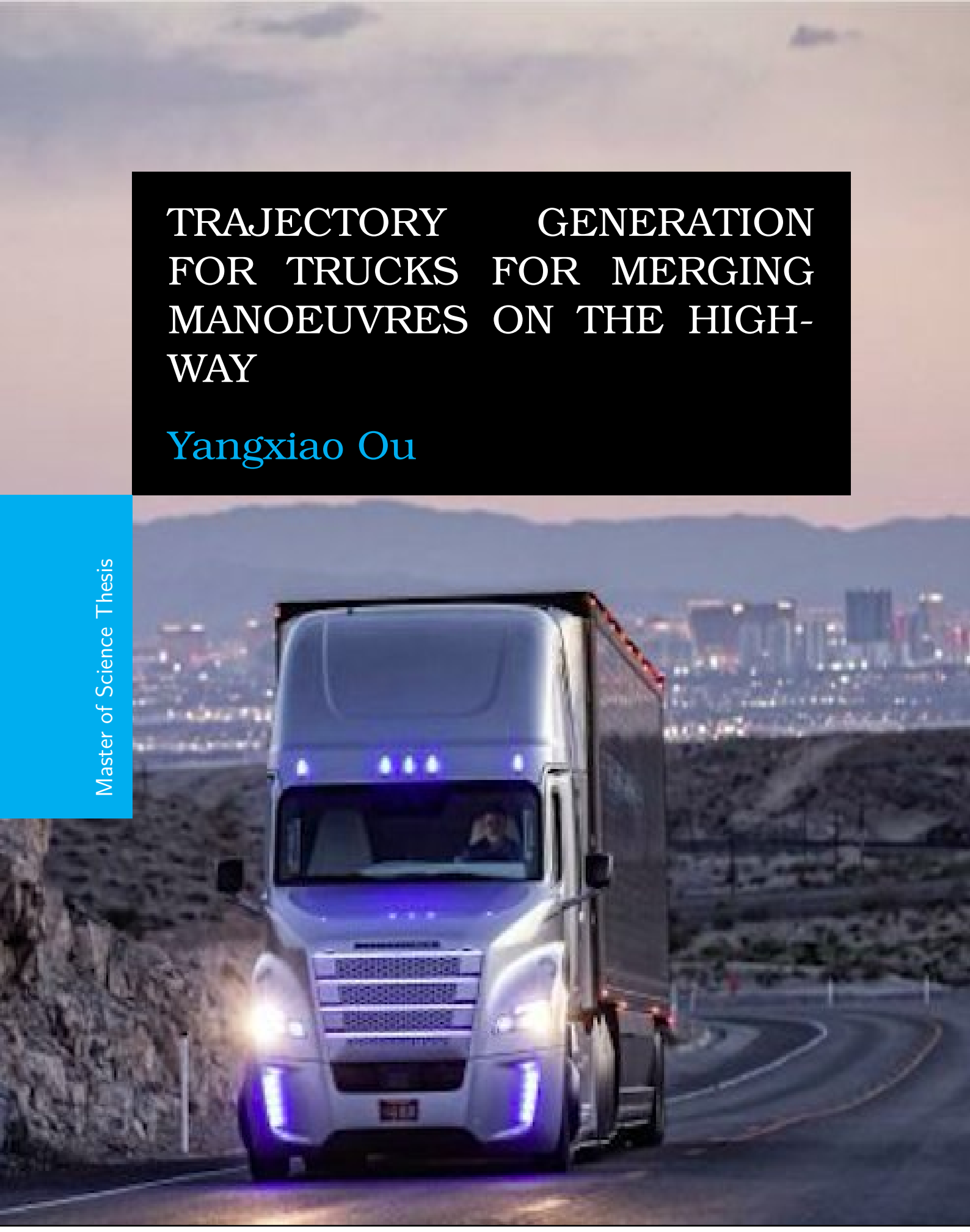
TRAJECTORY GENERATION FOR TRUCKS FOR MERGING MANOEUVRES ON THE HIGH-WAY

Yangxiao Ou

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

TRAJECTORY GENERATION FOR TRUCKS FOR MERGING MANOEUVRES ON THE HIGHWAY

Master of Science Thesis

For the degree of Master of Science in Mechanical Engineering at Delft University of Technology

Yangxiao Ou

May 18, 2016

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

TRAJECTORY GENERATION FOR TRUCKS FOR MERGING MANOEUVRES ON THE HIGHWAY

by

Yangxiao Ou

in partial fulfillment of the requirements for the degree of

Master of Science Mechanical Engineering

Dated: May 18, 2016

Supervisor(s): _____

Prof.dr.ir. Bart De Shutter
Dr.ir. Riender Happee
Dr.ir. Meng Wang

Reader(s): _____

Prof.dr.ir. Bart De Shutter
Dr.ir. Riender Happee
Dr.ir. Meng Wang
Dr.ir. A.J.J. van den Boom

Abstract

Trucks are largely used in road transportation worldwide in recent decades, and make great contributions to the GDP [1]. Statistics shows that a large number of truck-involved accidents occur related to the ramp in merging manoeuvres on the highway [2]. The accidents involved in trucks can lead to a considerable economic cost [3]. The development of automated vehicles largely prompts the growth of vehicle active safety. In this thesis, we would like to address the problem of trajectory generation of trucks in merging manoeuvres.

The trajectory generator is an important autonomous sector of a fully automated vehicle. A good generator can select the right moment for lane-changing manoeuvres by identifying a suitable gap in the traffic at the target lane, and define the path and acceleration profile. The potential algorithm, the optimisation-based algorithm and the sample-based algorithm are the main approaches to solve the motion planning problem. Due to its obvious drawbacks, the potential algorithm is not frequently used for the vehicle trajectory generation. The optimisation-based algorithm can get very accurate solutions; however, its performance depends on the development of the solver since most of the available solvers cost much when handling with nonlinear problems. The sample-based algorithm is a flexible method, and can be modified to fit a large range of situations.

The sample-based algorithms are widely used in motion planning of small car-like robots. A lot of efforts have been paid on the studies of the sample-based algorithm. One of the most famous sample-based algorithms is the Rapidly-exploring Random Tree (RRT) algorithm. It is an algorithm that can cover the entire configuration space, and select a best path from the start state to the goal state quickly. However, the RRT algorithm is mostly used on small car-like robots with relatively low speed. Moreover, the results obtained by the RRT algorithm are usually not accurate. To address this problem, some modified RRT algorithms have been proposed. But most of the modified RRT algorithms have more complex structures, and achieve an accurate solution at the cost of more computational cost.

The kinodynamic planning problem is to find a motion that goes from a start state to a goal state while satisfying all constraints of a nonlinear system. Trajectory generation of

trucks is actually a kinodynamic planning problem. In this thesis, we would like to develop an RRT algorithm to solve the path planning problem of a large truck for merging manoeuvres on the highway. The basic RRT algorithm cannot solve the trajectory generation problem alone, since the basic RRT algorithm only focuses on the propagation of the tree and does not take the system's dynamic into consideration. It should be incorporated with other algorithms to solve the path planning problem.

The trajectory generated by the RRT algorithm cannot satisfy all constraints of the vehicle system. To solve this problem, a dynamic model that includes non-linear tyre properties with limits on longitudinal acceleration and steering angle, and with first order driveline and steering dynamics is incorporated in the RRT algorithm. Some strategies are incorporated in the RRT algorithm to improve the performance. The sample bias is introduced to the RRT algorithm to reduce the number of the waste samples by increasing the sampling probability of the nodes near to the goal region. The node selection method increases the choices of the nodes for the tree and reduces the sampling times. This method works by solving the non-linear system for multiple times before adding the resulted node to the tree. The combination of criteria strategy uses different criteria to stop the algorithm when the goal is far away and is close. This method improves the efficiency of the algorithm.

The RRT algorithm is developed for both online and offline implementation. The offline RRT algorithm is developed and implemented over the open-loop system and the closed-loop system. The simulation results show that a trajectory planned by the closed-loop RRT algorithm shows less disturbance and is smoother than a trajectory planned by the open-loop RRT algorithm. The real-time implementation is realised by updating the offline planning algorithm over closed-loop system. The replanning algorithm with regard to the highway situation is developed in this work.

Collision avoidance is incorporated with the RRT algorithm by applying the intersection algorithm on the bounding volumes, since the intersection algorithm is a simple and quick method to detect the collision. The surrounding traffic is modelled as a set of Axis-aligned Bounding Boxes, and the truck is modelled as an Oriented Bounding Boxes in the configuration space. The intersection algorithm is applied on these bounding volumes to check the intersection of them.

This work has developed a real-time RRT algorithm for trajectory generation for trucks for merging manoeuvres on the highway. We implement the RRT algorithm on both the open-loop system and the closed-loop system, propose ideas to improve the algorithm, and discuss the parameters that affect the algorithm in this work. It is a piece of work for the application of the RRT algorithm on the planning problems for large vehicles.

Table of Contents

Acknowledgements **xi**

1 Introduction **1**
1-1 Problem Statement . . . 2
1-2 Thesis Contributions . . . 3
1-3 Overview of the Thesis . . . 4

2 Modelling **5**
2-1 Coordinate Systems . . . 5
2-2 Modelling of Vehicle . . . 7
2-2-1 The Merging Manoeuvre . . . 7
2-2-2 The Truck Model . . . 8
2-2-3 Parameters of the Truck Model . . . 11
2-3 Modelling of Surrounding Obstacles . . . 13
2-4 Summary . . . 15

3 RRT Algorithm for Trajectory Generation **17**
3-1 The Basic RRT Algorithm . . . 17
3-2 Kinodynamic Motion Planning . . . 20
3-3 Strategies for the RRT Algorithm . . . 21
3-3-1 Sampling Strategy . . . 21
3-3-2 Node Connection Strategy . . . 23
3-3-3 Algorithm Stop Strategy . . . 24
3-4 Offline Trajectory Generation . . . 25
3-4-1 Offline Planning over Open-Loop Dynamics . . . 25
3-4-2 Offline Planning over Closed-Loop Dynamics . . . 27
3-5 Real-Time Trajectory Generation . . . 29

3-6 Obstacle Avoidance . . . 31
3-6-1 Bounding Volumes . . . 32
3-6-2 Intersection Detection . . . 33
3-6-3 Collision Avoidance on the Highway . . . 34
3-7 Summary . . . 37

4 Controller Design **39**
4-1 The Control Architecture . . . 39
4-2 Speed Controller . . . 40
4-2-1 PID Controller . . . 41
4-2-2 PID Controller Design Method . . . 41
4-2-3 PID Speed Controller Design . . . 42
4-3 Steering Controller . . . 45
4-3-1 Pure-Pursuit Control Theory . . . 45
4-3-2 Modified Pure-Pursuit Steering Controller . . . 48
4-3-3 Steering Controller Design for the Truck . . . 51
4-4 Summary . . . 55

5 Simulations and Results **57**
5-1 Parameterisation of Bias . . . 59
5-2 Offline Implementation . . . 64
5-2-1 Offline Implementation over the Open-Loop System . . . 64
5-2-2 Offline Implementation over the Closed-Loop System . . . 65
5-2-3 Discussion . . . 68
5-3 Real-Time Implementation . . . 71
5-3-1 Real-Time Implementation in Different Scenarios . . . 72
5-3-2 Discussion . . . 80
5-4 Summary . . . 83

6 Conclusions and Recommendations for Future Work **85**
6-1 Discussions and Conclusions . . . 85
6-2 Recommended Future Work . . . 86

A Appendix **89**
A-1 MATLAB Code . . . 89
A-1-1 Offline RRT Algorithm over Open-Loop System . . . 89
A-1-2 Offline RRT Algorithm over Closed-Loop System . . . 91
A-1-3 Online RRT Algorithm over Closed-Loop System . . . 94
A-1-4 RRT Template . . . 96
A-1-5 Runga-Kutta Algorithm . . . 105
A-1-6 Vehicle Model . . . 105
A-2 SIMULINK Model . . . 106

Glossary **111**
List of Acronyms . . . 111
List of Symbols . . . 111

List of Figures

2-1 The body-fixed coordinates and forces affecting a vehicle. This figure is from [4] 6

2-2 The configuration of the highway entrance. This picture is adopted from Janson (1998). 7

2-3 The geometric description of the merging manoeuvres. The yellow blocks are the putative follower (PF) vehicle and the putative leader (PL) vehicle, respectively. The blue block is the truck intended to merge into the highway. 8

2-4 The configuration of a truck. CoG of the centre of the gravity. L is the length of the wheelbase, L_f is the distance front axle to CoG, L_r is the distance rear axle to CoG, F_{z1} is the front axle load, and F_{z2} is the rear axle load. 8

2-5 The vehicle model with Ackerman steering. 9

2-6 The cornering stiffness and vertical load. This figure is adopted from [4] 11

2-7 Space-time representation of large truck preparing to merge into a lane. 13

2-8 The geometric description of a dynamic obstacle in the highway. The yellow block is a dynamic obstacle, the blue block is the truck intended to merge into the highway, and the grey blocks are the other dynamic obstacles on the highway. . 14

2-9 The geometric description of Staggered car-following behaviour on the highway. The yellow block is the putative follower vehicle, the blue block is the truck intending to merge into the highway, and the orange block is the putative leader vehicle in the target lane. 15

3-1 Construction of an RRT for $X = [0, 100] \times [0, 100]$, $\triangle x = 1$, and $x_{init} = [50, 50]$, from http://msl.cs.uiuc.edu/rrt/about.html 19

3-2 An example of building a roadmap between the start state and a given goal state. The roadmap extends by sampling existing nodes and sampling the input vector u 20

3-3 An example of a finding trajectory. After the goal state is reached, the algorithm will backtrack the roadmap to find the best trajectory 21

3-4 The Gaussian distribution. The values less than one standard deviation away from the mean account for 68.27% of the set of the entire points in the configuration space; while two standard deviations from the mean account for 95.45%; and three standard deviations account for 99.73% 22

3-5 An example of the node selection in real-time planning. The yellow nodes are the potential nodes to insert into the tree. The dotted line is the potential path. . . 23
3-6 The configuration of the planner with open-loop prediction system. Given a sequence of inputs signals u, and then collect the output x to the tree when x is in the feasible space. 26
3-7 The configuration of the planner with a closed-loop system. Given a reference command r, the controller will generate commands u to system. 28
3-8 The flow chart of the implementation of the real-time RRT algorithm 30
3-9 Different types of bounding boxes. 32
3-10 Modelling the truck as a sphere in a configuration space when checking the collision. 35
3-11 Modeling the truck as an OBB in a configuration space when checking the collision. 35
3-12 Modeling the truck as an AABB in a configuration space when checking the collision. 35
3-13 The tail space attached on the the rear of the vehicles, the black blocks are the tail spaces. 36
3-14 The posture of the truck in the configuration space. The (X_c, Y_c) is the coordinate of the CoG in the inertial coordinate system, the θ_c is the heading angle when the truck's CoG located in (X_c, Y_c). 37

4-1 The overview of the control system of the autonomous vehicle. 40
4-2 The flowchart of a speed controller 40
4-3 The speed control loop 43
4-4 The step response of the closed-loop system with parameters $K_p = 1$, $K_i = 0$ and $K_d = 0$. 43
4-5 The responses comparison of the system with the PI controller and PID controller with different value of K_d. The data is obtained with $K_p = 19.33$ and $K_i = 72.5$, and $K_d = 0.5, 1, 3, 5, 8$. 44
4-6 The responses of the system with the PI controller. The values of V_{ref} are 12 m/s, 15 m/s,18 m/s, 20 m/s, 25 m/s. $K_i = 72.5$, $K_p = 19.33$. 45
4-7 The steering control loop. 45
4-8 The strategy of the pure-pursuit algorithm. 46
4-9 The response of the controller with look-ahead distances $L_d = 20\ m$, $L_d = 30\ m$, $L_d = 40\ m$,$L_d = 50\ m$, $L_d = 60\ m$. At first time, the vehicle is not on the path, and then tries to attain the ref path (the red dotted line). 47
4-10 The response of the controller with look-ahead distances $L_d = 20\ m$, $L_d = 30\ m$, $L_d = 40\ m$,$L_d = 50\ m$, $L_d = 60\ m$. At first time, the vehicle is on the path, and then tries to maintain the ref path (the red dotted line). 48
4-11 The geometric expression of the modified pure-pursuit algorithm. All angles and the lengths in this figure are positive by definition. 49
4-12 The straight reference path along the x-axis. 50
4-13 The performance of the steering controller with different values of L_d. 52
4-14 The performance of the controller with the modified pure-pursuit algorithm. The look-ahead distances L_d is chosen as $L_d = 30\ m$. The velocity $V = 18\ m/s$. The blue line is the reference curve, the red line is the tracking curve. 53
4-15 The performance of the controller with the modified pure-pursuit algorithm. The look-ahead distances L_d is chosen as $L_d = 30\ m$. The velocity $V = 13\ m/s$. The blue line is the reference curve, the red line is the tracking curve. 54

5-1 The road in the configuration space. The width of each lane is 3.5 m and the total length of the road is 450 m. The grey area is the infeasible area that the truck cannot enter into, and the white area is the drivable area. 57

5-2 The merging scenario of a truck. The orange blocks represent the putative leader vehicle and the putative follower vehicle. The dark blue block represents the truck. The red stars represent the start point and the goal point. The red dotted block is the goal region. 59

5-3 The configuration space used in this sections. The gaps between two vehicles are 100 m in the Lane 1, and are 150 m in the lane 2 in this surface. The dark blue block represents the truck, the light blue blocks present the surrounding traffics. 60

5-4 The distribution of 500 points with different standard deviation σ_r and σ_θ, the means μ of the Gaussian distributions are $\mu = 0$. 62

5-5 The distribution with the parameters listed in Table 5-2. 500 samples have been plotted in this figure. The goal point is $(210, 6.25)$ 62

5-6 The generated path and its corresponding velocity, steering angle, orientation angle, steering rate and acceleration. The step-size $\Delta T = 0.03\ s$. 65

5-7 The generated path and its corresponding velocity, steering angle, orientation angle, steering rate and acceleration. The step-size $\Delta T = 0.05\ s$. 66

5-8 The generated trajectory. This trajectory is obtained with $V_{init} = 16.7 m/s$ (ca. 60km/h). 67

5-9 The reference path (the red line in this figure) and the corresponding generated path (the blue line in this figure). 67

5-10 The planned trajectory (the orange line) and the reconstructed trajectory (the yellow line). 68

5-11 The generated trajectory. This trajectory is obtained with combined stop criteria. 69

5-12 The reference path (the red line), and the planned trajectory (the blue line). This trajectory is obtained with combined stop criteria. 69

5-13 The merging scenario of a truck from the ramp to the highway in the real-time implementation. The initial speed of the truck is 16.7 m/s (ca.60 km/h). The speed of the vehicles in the target lane is 18.3 m/s. 71

5-14 The planned path, orientation angle, velocity, steering angle, acceleration and steering rate. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05$ s, $gap = 100$ m, $a = 0\ m/s^2$, and truck initial speed is $V = 16.7\ m/s$ (ca. 60 km/h). 72

5-15 The planned path, orientation angle, velocity, steering angle, acceleration and steering rate. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05\ s$, $gap = 100\ m$, $a = 0\ m/s^2$, and truck initial speed is $V = 16.7\ m/s$ (ca.60 km/h). 74

5-16 The planned path, reference paths and reconstructed path. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05\ s$, $gap = 100\ m$, $a = 0\ m/s^2$, and truck initial speed is $V = 16.7\ m/s$ (ca.60 km/h). 74

5-17 The central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle, respectively. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a = 0\ m/s^2$, $\Delta t = 0.05\ s$, $gap = 100\ m$, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h). 75

5-18 The planned path, orientation angle, velocity, steering angle, acceleration and steering rate. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05$ s, $gap = 50$ m, $a = 0$, and truck initial speed is $V = 16.7\ m/s$ (ca.60 km/h). 75

5-19 The planned path, reconstructed path and the reference path. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a = 0$, $\Delta t = 0.05$ s, $gap = 50$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h). 76

5-20 The central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle, respectively. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a = 0$, $\Delta t = 0.05$ s, $gap = 50$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h). . 76

5-21 The planned path, orientation angle, velocity, steering angle, acceleration and steering rate. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05$ s, $gap = 50$ m, $a_{PF} = 0.5\ m/s^2$, and truck initial speed is $V = 16.7\ m/s$ (ca.60 km/h). 77

5-22 The planned path, reconstructed path and the reference path. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a_{PF} = 0.5\ m/s^2$, $\Delta t = 0.05$ s, $gap = 50$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h). . 78

5-23 The central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle, respectively. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a_{PF} = 0.5$ m/s^2, $\Delta t = 0.05$ s, $gap = 50$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h). 78

5-24 The planned path, orientation angle, velocity, steering angle, acceleration and steering rate. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05$ s, $gap = 50$ m, $a_{PL} = -0.5\ m/s^2$, and truck initial speed is $V = 16.7\ m/s$ (ca.60 km/h). 79

5-25 The planned path, reconstructed path and the reference path. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a_{PL} = -0.5$ m/s^2, $\Delta t = 0.05$ s, $gap = 50$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h). 79

5-26 The central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle, respectively. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a_{PL} = -0.5$ m/s^2, $\Delta t = 0.05$ s, $gap = 50$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h). 80

5-27 The planned path, orientation angle, velocity, steering angle, acceleration and steering rate. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05\ s$, $gap = 40\ m$, $a = 0\ m/s^2$, and truck initial speed is $V = 16.7\ m/s$ (ca.60 km/h). 81

5-28 The planned path, reconstructed path and the reference path. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a = 0\ m/s^2$, $\Delta t = 0.05$ s, $gap = 40$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h). . 81

5-29 The central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle in x-axis and y-axis, respectively. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a = 0\ m/s^2$, $\Delta t = 0.05$ s, $gap = 40$ m, and truck initial speed $V = 16.7$ m/s (ca.60 km/h). 82

A-1 SIMULINK Model. 106

List of Tables

2-1 The parameters of the truck. These values are adopted from [4]. 12

3-1 Timing of overlap testes for different bounding volumes. The AABB represents the axis-aligned bounding box, and the OBB represents oriented bounding box. These data are from [5]. 36

4-1 Ziegler-Nichols Method. 42

4-2 The effects of increasing one parameter K_i, K_p or K_d independently. This table is adopted from [7] 42

4-3 The minimum value of the look-ahead distance L_d with different velocities. . . . 52

5-1 Comparison the costs of the algorithm with bias and without bias. The costs are described by the needed time to achieve the goal, and the nodes in the tree. The tests are implemented with the same conditions. The planned paths for these five scenarios consist of nearly the same number of nodes. 60

5-2 Parameters for the bias to the distributions used in comparison between the algorithm with bias and without bias. 60

5-3 The comparison results of the costs with different value of r_0. $\sigma_\theta = \frac{\pi}{12}$, $\sigma_r = 3$, $\mu_r = 0$ and $\mu_\theta = 0$. 61

5-4 The comparison results of the costs with different value of σ_θ and σ_r. The $\mu_r = 0$ and $\mu_\theta = 0$. The symbol '–' means no nodes is generated. 63

5-5 Parameters for the bias to the distributions of merging maneuvers on the highway. 63

5-6 The computation time of the algorithm to find a feasible path with different time step sizes. The generated nodes means the nodes in the tree, and the nodes of path means the number of the nodes constructing the planned path. 65

5-7 Different scenarios used in the real-time implementation. 71

5-8 Results of different scenarios in this section. The state includes the coordinates of the trucks CoG and the orientation angle in this table. 82

Acknowledgements

First and foremost, I would like to thank my chair supervisor Prof. Dr. Ir. Bart De Schutter, and my advisor Dr. Ir. Riender Happee, for the chance of this challenging project and their suggestions on many professional problems, process management and academic writing.

Then I want to thank my daily supervisor Dr. Ir. Meng Wang for his assistance during this thesis work. I received his guidance and support in regular meetings and discussions. Thanks for his patience and encouragement. I have learned a lot from the meetings, the discussions and the feedback.

In addition, I would also like to thank Jun Zhang for his help on implementation problems, Barys Shyrokau for his valuable suggestions on vehicle modelling, Mukunda Bharatheesha for his professional advices on the sample-based path planning problem, and my friends who gave me direct or indirect input.

Finally, I am grateful to my parents for their selfless love and unconditional support throughout all my studies here, and to my fiancé for his understanding and confidence in me.

Yangxiao Ou
Delft, Febuary 2016

‘If A is a success in life, then A equals X plus Y plus Z. Work is X; Y is play; and Z is keeping your mouth shut.’

— *Albert Einstein*

Chapter 1

Introduction

Trucks are largely used in road transportation worldwide in recent decades, and make great contributions to the GDP [1]. In addition, trucks have the potential to decrease the transportation cost, to decrease traffic jams and to reduce the emissions (more freight per truck and driver). Hence, they are widely used on highways [8]. However, in recent years the number of accidents involving trucks has increased [9]. The accidents involving trucks can lead to a considerable economic cost [3]. Statistics show that a large number of truck-involved accidents are related to the on-ramp merging manoeuvres on highways [2]. Besides, they also show that on-ramp accidents occupy larger percentage than off-ramp accidents, and among all the on-ramp accidents the merge accidents share the largest percentage. It is meaningful to improve the safety of trucks on the ramps of highways in merging manoeuvres.

Nowadays automated vehicles have become the new power and hot spot of the growth of the automobile industry. The development of automated vehicles largely prompts the growth of vehicle active safety. A trajectory generator is one of the most important automated sectors of the automated vehicle. The task of the trajectory generator is to plan a trajectory for the vehicle. A good generator can select the right moment for the lane-changing manoeuvres by identifying a suitable gap in the traffic at the target lane, and define the path and acceleration profile.

There are three main approaches to solve the trajectory generation problem: the potential method, the optimisation-based method and the sample-based method. Due to its limitation of implementation in dynamic environment, the potential method is seldomly used for solving the trajectory generation problem now. In recent years, the optimisation-based method and sample-based method have received a lot of attention. The optimisation-based method can solve the trajectory generation problem accurately, but it is too sensitive to modelling, and brings much computational complexity in real-time implementation.

The sample-based method is based on a family of sample-based algorithms. The most influential sample-based algorithms for trajectory generation include Probabilistic Road Map (PRM) and Rapidly-exploring Random Tree (RRT) [10]. These two algorithms have the same

idea of connecting nodes sampled randomly from the configuration space; however, they differ in the method of constructing a graph to connect these nodes. The PRM algorithm [11] is a multiple-query method that constructs a roadmap with a rich set of collision-free trajectories first, and then connects the initial state with a final state in this roadmap by computing a cost-efficient path. It is valuable in highly structured environments, but it does not work well in most real-time planning problems due to computational challenges. For the online motion planning problems, the RRT algorithm has obvious advantages [12], for instance, it does not need to set the number of iterations in advance, and returns a solution as soon as the desired trajectory is found, which enables real-time implementation. Furthermore, the RRT algorithm does not require connecting two nodes exactly and can more easily deal with differential constraints. Since it is a single-query method, which represents that every node in the tree has only one parent node, the computational efforts of RRT are relatively low. Hence the RRT algorithm are widely used in the motion planning for robotic vehicles.

In the past, most research has been focused on using sampled-based algorithms to plan trajectories for small-sized robotic cars with low speed. In this work, we would like to develop the RRT algorithm for trajectory generation for trucks for merging manoeuvres at ramps on the highway.

1-1 Problem Statement

Truck-involved accidents lead to a large economic cost. Statistics shows that a large number of truck-involved accidents are related to the ramp in merging manoeuvres on the highway, and among all the ramp-related accidents the merge accidents share the largest percentage. To prevent the accidents involving trucks for merging manoeuvres, we would like to develop an algorithm to generate a feasible and collision-free trajectory for trucks to complete merging manoeuvres on the highway.

The goal of trajectory generation for vehicles is to find a feasible path with corresponding velocity from the initial state to a final state with consideration of a set of linear/non-linear constraints. A trajectory differs from a path. The path represents a sequence of positions defined in a configuration space, whereas a trajectory equals a path plus the velocity along it. Arguably, the motion planning problem for a vehicle is equal to solve the optimisation problem:

$$\min_{u} \quad \int_0^{t_f} \Phi(x(t), u(t))dt$$

subject to non-linear system:

$$\dot{x} = f(x(t), u(t)), \quad x(0) = x_0, \quad x(t_f) = x_{t_f}$$

where $x(t) \in \mathbb{R}$ is the state, $x_0 \in \mathbb{R}$ is the initial state at time $t = 0$, and $u(t) \in U$ is the input vector, subject to constraints:

$$u(t) \in U, \quad x(t) \in X_{free}(t), \quad t \in [t_0, t_f]$$

where t_f is limited in the range $[0, \infty)$, and X_{free} represents the collision-free space. For offline planning, it is time-independent, whereas for online planning, X_{free} changes with time.

The research goal of this thesis is to develop a real-time algorithm for trajectory generation for trucks in merging manoeuvres on the highway. Previous work has shown promising results for path planning for car-like robots using RRT algorithm in the low speed condition. In this work, we would like to develop the RRT algorithm for the path planning in high speed scenarios. The RRT algorithm cannot solve the trajectory generation problem individually. It needs to combine with other methods to plan a satisfactory trajectory. The RRT algorithm has its own drawbacks that may lead the algorithm failing to work or performing badly. In this thesis, we would like to improve the performance of the RRT algorithm in the trajectory generation as well.

The main research problems are: modelling the truck, road and surrounding traffic; develop offline and online algorithms for generating trajectories of the truck; develop a collision detection algorithm incorporated in the trajectory generation algorithm; and develop a control method for the truck.

1-2 Thesis Contributions

The Rapidly-exploring Random Tree algorithm (RRT) is mainly applied on small car-like robots in the low speed situation. The work in this thesis implements an existing state-of-art RRT algorithm to a large size car-like robot like the truck in the highway situation. A trajectory generator is shown to be able to generate collision-free trajectory for a controlled model of the truck in merging manoeuvres on the ramps of the highways.

The discussion of the parameters that influence the results of the RRT algorithm provides good guidances of the future development and implementation of RRT algorithm in trajectory generation. Moreover, the strategies for the application of the RRT algorithm and the development of the combined stop criteria for the RRT algorithm give directions to improve the algorithm's performance in trajectory generation in typical manoeuvres.

The intersection algorithm has been incorporated in the RRT algorithm in this work for collision avoidance is incorporated by employing the bounding volumes. The proposal of the extra tail for vehicles when detecting the collision provides a way to reduce the potential accidents caused by the inertia of vehicles.

The offline implementation shows different performances of the RRT algorithm over open-loop system and closed-loop system in trajectory generation for merging manoeuvres, and provides an evidence that the closed-loop RRT algorithm has advantages over the open-loop RRT algorithm in trajectory generation. The real-time implementation enables future related research on the same type in MATLAB/SIMULINK environment, and presents the replanning algorithm for merging manoeuvres in the highway situation.

This work presents a proof-of-concept for the application for the closed-loop RRT algorithm of merging manoeuvres for trucks on the highways.

1-3 Overview of the Thesis

In Chapter 2 modelling of the vehicle, road and surrounding traffic is introduced. The merging manoeuvres are illustrated at the beginning of this chapter. In the middle, a dynamic model including non-linear tyre properties with limits on longitudinal acceleration and steering angle, and with first order driveline and steering dynamics is presented. The parameters of a truck are determined. At last, the surrounding traffic is modelled.

Chapter 3 focuses on the solution of the trajectory generation problem. It is the main contain of the thesis. It gives a simple view of the basic RRT algorithm, and proposes some useful strategies for the application of the RRT algorithm in path planning. It presents the offline RRT algorithm over the open-loop system and over the closed-loop system. After this, the online replanning algorithm for merging manoeuvres is developed. The collision avoidance method is also introduced in this chapter.

The topics of Chapter 4 are the controllers that are used in the motion planning problem. Two control algorithms are introduced. In this chapter, a PID controller is designed to control the speed, and a pure-pursuit controller is designed to control the steering of the vehicle.

Chapter 5 focuses on the results of the offline and online RRT algorithm implementations in merging manoeuvres. Both the open-loop system and closed-loop system are used in the offline trajectory generation, and a comparison of the results is made. The real-time trajectory generation is applied in different scenarios, and the results obtained by two different stop criteria are compared.

Finally, in Chapter 6 the conclusions and contributions of the thesis are summarised, and recommendations for future work are listed.

Chapter 2

Modelling

In classical mechanics, the parameters that define the configuration of a system are the generalised coordinates, and the vector space defined by these coordinates is the configuration space of the physical system [13]. The RRT algorithm works in a configuration space; therefore, all the information in the real world needs to be modelled into the configuration space.

In this chapter, we would like to model the information of the vehicle and surrounding obstacles into the configuration space. Before introducing the modelling of the vehicle and the obstacles, we would like to introduce the coordinate system first. The knowledge of the coordinate system can help us to analyse the problem well. Then a vehicle model is illustrated in this chapter. At the end of this chapter, the surrounding obstacles are modelled in the configuration space.

2-1 Coordinate Systems

In trajectory generation problems of a ground vehicle, the inertial coordinate system and the body-fixed coordinate system are intensively used in planning and analysis. The inertial coordinate system is used to describe the location of vehicles in a plane. The axes X, Y and Z are considered to be neither accelerating nor rotating, and thus there is an inertial frame in which Newton's laws are easily written. The location of a vehicle can be presented in an inertial system by the coordinate (X, Y, Z) of its mass centre point.

It seems reasonable to use an inertial coordinate system to describe the dynamics of a vehicle, but in fact, a moving coordinate system attached to the vehicle itself is more preferred by analysers. The body-fixed coordinate system and the major forces affecting a vehicle are shown in Figure 2-1. In this body-fixed coordinate system, the forward motion of a vehicle is described in the positive $x - axis$, the lateral motion is represented in the positive $y - axis$, and the vertical motion is described by $z - axis$ whose direction is determined by the right-handed rule. In addition, the rotation of a vehicle is also considered in this coordinate system.

The yaw rotation is around the $z-axis$, the pitch rotation is around the $y-axis$, and the roll rotation is around the $x-axis$.

Figure 2-1: The body-fixed coordinates and forces affecting a vehicle. This figure is from [4]

The body-fixed coordinate system of a vehicle can be transferred into the inertial coordinate system by using the rotation matrix. When the vehicle only rotates in the body-fixed coordinate system S_B, with a yaw angle ψ, around the $z-axis$ of the inertial coordinate system S_I, the rotation matrix $R_{I,z}$ is:

$$R_{I,z} = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2-1}$$

In addition, when the vehicle body rotates with an angle in the body-fixed coordinate system S_B, with a roll angle γ, around the $x-axis$ of the inertial coordinate system S_I, the rotation matrix $R_{I,x}$ is:

$$R_{I,x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\gamma) & -sin(\gamma) \\ 0 & sin(\gamma) & cos(\gamma) \end{bmatrix} \tag{2-2}$$

Finally, when the vehicle body rotates with an angle in the body-fixed coordinate system S_B, with a pitch angle ζ, around the $y-axis$ of the inertial coordinate system S_I, the rotation matrix $R_{I,y}$ is:

$$R_{I,y} = \begin{bmatrix} cos(\zeta) & 0 & sin(\zeta) \\ 0 & 1 & 0 \\ -sin(\zeta) & 0 & cos(\zeta) \end{bmatrix} \tag{2-3}$$

In this section, we are focused on the coordinate systems. There are two different coordinate systems used in vehicle path planning analysis: the inertial coordinate system, and the body-fixed coordinate system. The body-fixed coordinate system can be transferred to the inertial

coordinate system by using the rotation matrix. In this project, we define the configuration space in the $X-Y$ plane of the inertial coordinate system. In the next section, we would like to model the truck in the inertial coordinate system.

2-2 Modelling of Vehicle

The previous section has introduced the information about the coordinate systems. In this section, we would like to address the modelling problem of the truck in the coordinate systems. Before introducing the model used in the planner, we describe the merging manoeuvres firstly. In the middle, a truck model for solving the dynamics of the truck is depicted. At the end of this section, the parameters of the truck model are listed.

2-2-1 The Merging Manoeuvre

Before modelling the vehicle in the configuration space, we would like to present the merging manoeuvres of the truck first.

The configuration of a double-lane highway entrance is represented in Figure 2-2. It contains three sections: the area of the upstream, the ramp connection area, and the area of the downstream. The length of the area of the upstream is about 0.25 miles (c.a. 402 m). The length of the ramp connection area is about 0.219 miles (c.a. 352 m) in average. The length of the area of the downstream is about 0.15 miles (c.a. 240 m). The width of each lane is 3.5 m. We assume that the sensors can detect the road information in a range of 180 m. This is, the truck can detect all the traffic information in the ramp connection area.

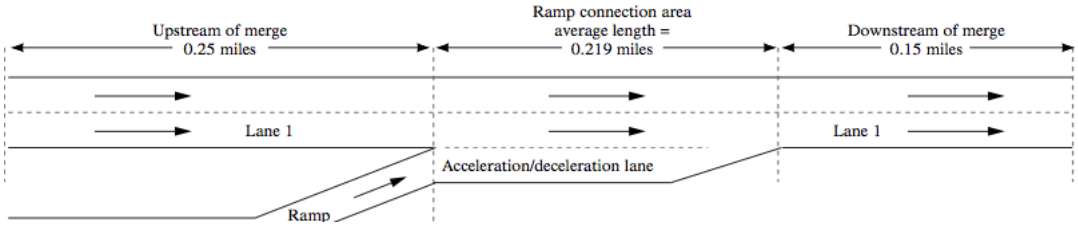


Figure 2-2: The configuration of the highway entrance. This picture is adopted from Janson (1998).

The merging manoeuvres can be described as follows (see Figure 2-3). The truck intends to enter the highway from the ramp through the acceleration lane to the target lane (Lane 1 in the Figure 2-2). It finds an acceptable gap at first, and then steers to merge into this gap. In order to avoid the unexpected collisions and to complete the lane-changing manoeuvres, the truck needs to accelerate to achieve a similar velocity with the putative leader (PL) vehicle and/or the putative follower (PF) vehicle at the same time. After entering the highway, the truck will follow the leader vehicle in the same lane.

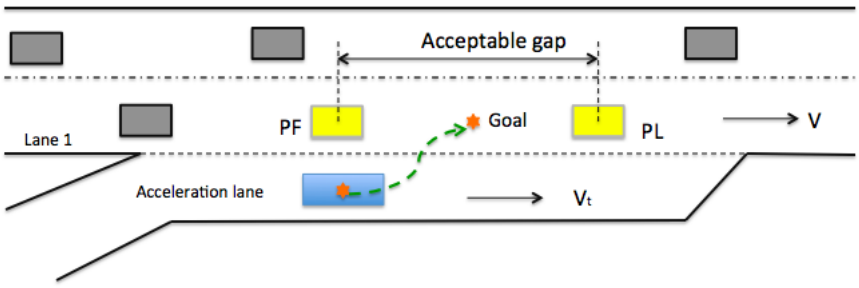


Figure 2-3: The geometric description of the merging manoeuvres. The yellow blocks are the putative follower (PF) vehicle and the putative leader (PL) vehicle, respectively. The blue block is the truck intended to merge into the highway.

2-2-2 The Truck Model

After introducing the merging manoeuvres, we represent the truck model that will be used in the motion planner. The truck, as the Figure 2-4 show, is a motor vehicle designed to transport the cargo. Trucks vary greatly in size, power, and configuration, with the smallest being mechanically similar to an automobile. Medium trucks are usually defined as weighting between 6000 kg and 15000 kg. Due to their large size and inertia, they show slow behaviour when merging into the highway.

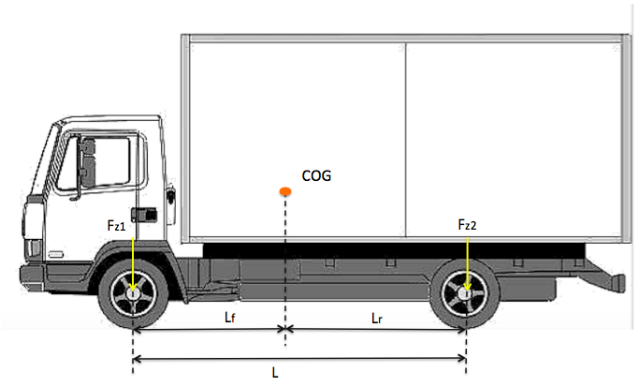


Figure 2-4: The configuration of a truck. CoG of the centre of the gravity. L is the length of the wheelbase, L_f is the distance front axle to CoG, L_r is the distance rear axle to CoG, F_{z1} is the front axle load, and F_{z2} is the rear axle load.

To simplify the problem, we assume that the truck moves in a 2-dimensional plane, and the influence of the roll and pitch can be ignored. That is, we only consider the movement of the truck in the $X-Y$ plane. Hence, the truck configuration space can be described by the global position and orientation variables, denoted as (X, Y, θ). Besides, we regard the truck as a rigid body, as well as the tires. Based on these assumptions, the popular Ackerman (or bicycle) model depicted in Figure 2-5 can be employed to approximate the vehicle motion. The Ackerman model assumes that the two front wheels turn slightly differentially. Thus, the

instantaneous rotation centre can be purely computed by kinematic means.

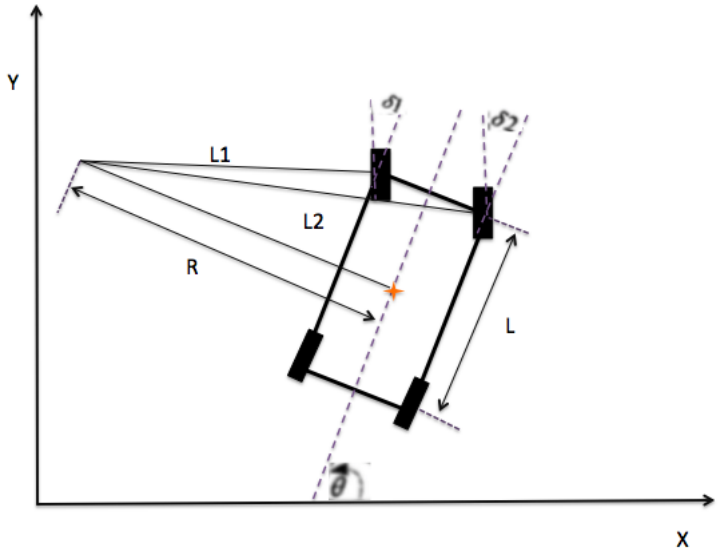


Figure 2-5: The vehicle model with Ackerman steering.

Let κ denote the instantaneous curvature of the trajectory, and we can obtain Equation 2-4:

$$\kappa = \frac{1}{R} = \frac{2tan(\delta)}{L} = \frac{d\theta}{ds} \tag{2-4}$$

where R is the radius of curvature, L the wheelbase, δ is the steering angle, s is the path length, and θ is the vehicle orientation in an inertia frame. The dynamics of θ can be obtained by Equation 2-5:

$$\dot{\theta} = \frac{d\theta}{dt} = \frac{d\theta}{ds}\frac{ds}{dt} = \kappa V = \frac{2V}{L}tan(\delta) \tag{2-5}$$

where V is the velocity of the vehicle's CoG. The dynamics of X and Y are depicted in Equation 2-6 and Equation 2-7:

$$\dot{X} = \frac{dX}{dt} = Vcos(\theta) \tag{2-6}$$

$$\dot{Y} = \frac{dY}{dt} = Vsin(\theta) \tag{2-7}$$

Taking the dynamics of the velocity V and the steering angle δ into consideration, we can get the kinematic model as depicted in Equation 2-8:

$$\begin{aligned} \dot{X} &= Vcos(\theta) \\ \dot{Y} &= Vsin(\theta) \\ \dot{\theta} &= \frac{2V}{L}tan(\delta) \\ \dot{V} &= a \\ \dot{\delta} &= r \end{aligned} \tag{2-8}$$

where a is the acceleration and r is the yaw rate.

The kinematic model is frequently used in solving path-planning problems. It is easy to understand, and is able to capture the location information of the vehicle in a configuration space. In addition, solving this kind of model needs low computational costs due to its simplification. In the real-time planning problem, a lower computational burden means more efforts to optimise the planned path, hence a better trajectory can be found.

However, this kinematic model does not take the skid of the vehicle into consideration. It is restrictive and does not allow considering neither skidding when turning at high speed nor the slip angle. Hence, when use this kind of model in a planner, it means lots of approximations. A path planned with this kind of model needs corrections to guarantee the safety. In order to capture the effect of the side slip, we can introduce a term G_{ss} described in the Equation 2-9 into the yaw rate $\dot{\theta}$ in Equation 2-8 [14]:

$$G_{ss} = \frac{1}{1 + (\frac{V}{V_{char}})} \tag{2-9}$$

The term G_{ss} is a static gain of the yaw rate $\dot{\theta}$, and is determined by the characteristic velocity V_{char} [15]. The characteristic velocity can be calculated by Equation 2-10:

$$V_{char} = \sqrt{57.3\frac{Lg}{K}} \tag{2-10}$$

where L is the length of wheelbase, g is the gravity acceleration, and K is the understeer gradient.

The understeer gradient is a measure of how the steering needed for a steady turn changes as a function of lateral acceleration. It is one of the main measures for characterising steady-state cornering behaviour. The value of K is related to the cornering stiffness and the load of each tire. The value of the K can be calculated by Equation 2-11:

$$K = \frac{F_{z_f}}{C_{\alpha f}} - \frac{F_{z_r}}{C_{\alpha r}} \tag{2-11}$$

where F_z is the load on the tire and $C_{\alpha f}$ is the cornering stiffness.

There exists a response delay for the actuators, and this delay cannot be ignored, especially for a truck. The model in Equation 2-8 does not take the time delay into consideration. In this work, we choose a first-order lag term $\frac{1}{Ts+1}$ with time constant T in Laplace domain to model the delay of the actuators. We denote the input of the actuator as u_c and the output of the actuator as u. The dynamics of the actuator can be described by Equation 2-12 in Laplace domain and by Equation 2-13 in time domain. The parameters of the truck model will be addressed in the next subsection.

$$u = \frac{1}{Ts + 1}u_c \tag{2-12}$$

$$\dot{u} = \frac{1}{T}(u_c - u) \tag{2-13}$$

2-2-3 Parameters of the Truck Model

A fully loaded truck has longitudinal acceleration capabilities that are in the range of $[-g, 0.1g]$. Considering that active rollover mitigation systems kick in at about $0.4g$, the bound is suggested conservatively in the region of [2.5 m/s^2, 3 m/s^2]. Considering safety and comfort, in this case we choose a tight bound, [-2.5 m/s^2,1.5 m/s^2], for the acceleration. The steering actuation bound applies to the wheel angle δ and the wheel angular rate $\frac{d\delta}{dt}$. Steering compliance limits the angle to approximately 1 rad; on highway driving situation these values are typically much lower. In high-speed scenarios, the maximum steering angles would cause rollover, or understeer in case of a low friction road surface. Therefore, the road wheel angle is limited to 0.3 rad. The angular rate of the steering wheel angle, that is limited by the steering wheel actuator, is limited to 0.1 rad/s (c.a. 2.22 m/s^2 with regard to common speed of 80 km/h). As for a truck, we can choose $T_s = 1.5$ s and $T_a = 1.2$ s as the delay of the steering actuator and acceleration actuator, respectively [16].

In order to compute the characteristic velocity, we need the parameters of the truck. The parameters of the truck used in this thesis are listed in Table 2-1. According to Equation 2-11, we can obtain the value of the understeer gradient based on Table 2-1. The understeer gradient is related to the cornering stiffness. Figure 2-6 shows the cornering stiffness with respect to different loads. Based on Figure 2-6, we can determine the values of $C_{\alpha f}$ and $C_{\alpha r}$ with $C_{\alpha f} = 79.1$ N/rad and $C_{\alpha r} = 66.0$ N/rad. Then we can compute understeer gradient K, and get the value of K is equal to $K = 5.4$. We choose $g = 9.8$ m/s^2, and then we can calculate the characteristic velocity V_{char} based on Equation 2-10. The value of the characteristic velocity is $V_{char} = 22.8$ m/s.

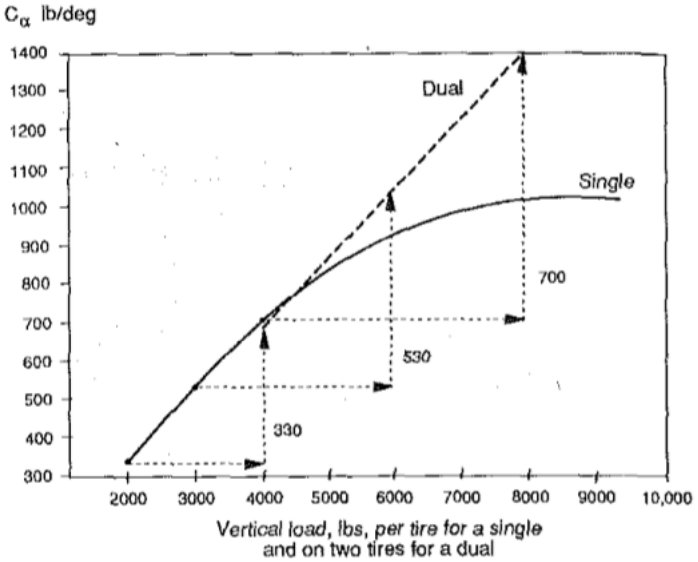


Figure 2-6: The cornering stiffness and vertical load. This figure is adopted from [4]

To summarise, the resulting vehicle model is represented in Equation 2-14, where (X, Y) is the coordinates in the inertia frame, θ is the orientation angle of the vehicle, V is the velocity

Symbol	Description	Unit	Value
L	length of wheelbase	m	5
F_{z_1}	front axle load	N	51659
F_{z_2}	rear axle load	N	22524
L_f	distance front axle	m	2.5
L_r	distance rear axle	m	2.5
W	width	m	2.5

Table 2-1: The parameters of the truck. These values are adopted from [4].

of the vehicle, L is the length of the wheelbase, a is the acceleration generated by the actuator to the motor, a_c is the acceleration sent to the actuator, δ is the steering angle generated by the steering actuator to the wheel, δ_c is the steering angle sent to the steering actuator. Actually, a_c and δ_c are the acceleration and steering angle generated by sampling the input space when the algorithm is implemented over the open-loop vehicle system. Moreover, the controllers generate them when the algorithm is applied on the closed-loop vehicle system.

$$
\begin{aligned}
\dot{X} &= Vcos(\theta) \\
\dot{Y} &= Vsin(\theta) \\
\dot{\theta} &= \frac{2V}{L} tan(\delta) G_{ss} \\
\dot{V} &= a \\
\dot{\delta} &= \frac{1}{T_s}(\delta_c - \delta) \\
\dot{a} &= \frac{1}{T_a}(a_c - a) \\
G_{ss} &= \frac{1}{(1 + \frac{V}{V_{char}})} \\
T_a &= 1.2\ s \\
T_s &= 1.5\ s \\
V_{char} &= 22.8\ m/s \\
-2.5\ m/s^2 \leq a &\leq 1.5\ m/s^2 \\
-0.3\ rad \leq \delta &\leq 0.3\ rad \\
-0.1\ rad/s \leq \dot{\delta} &\leq 0.1\ rad/s
\end{aligned}
\tag{2-14}
$$

The truck model depicted in Equation 2-14 will be used in the motion planner. During the merging manoeuvres, we need to consider both the truck and the vehicles around the truck on the highway. We have already modelled the truck in this section. In the next section, we would like to address the modelling problem of the surrounding vehicles.

2-3 Modelling of Surrounding Obstacles

The truck moves on the highway surrounded by some other vehicles. In order to implement a collision-free trajectory generation algorithm to the large truck with surrounding traffic, obstacles need to be modelled in the configuration space firstly. A clear distinction is made between a static obstacle and a dynamic obstacle, but a dynamic obstacle can be regarded as a special static obstacle in space-time. This is illustrated in Figure 2-7.

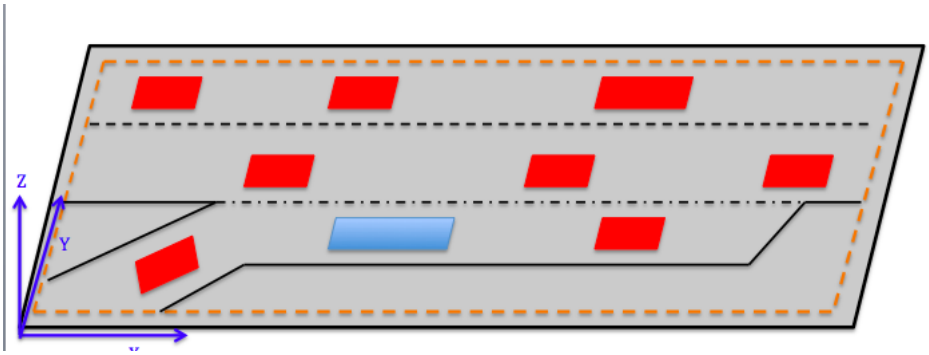


Figure 2-7: Space-time representation of large truck preparing to merge into a lane.

In merging manoeuvres, the static obstacles are road lines and road boundaries, whereas the dynamic obstacles are the surrounding vehicles. Each obstacle has three states:

1. Position on the geometric road map: $P_i(t)$
2. Velocity with regard to this geometric road map: $\dot{P}_i(t)$,
3. Information of the lane in which the vehicle is located: $Lane_i$

The states of static obstacles do not change with time. They will be marked as unfeasible area in the configuration space. The size of each static obstacle in the configuration space is decided by the largest contour area of the obstacle in the $X-Y$ plane. The states of the dynamic obstacles change with time. The dynamic obstacles in the merging manoeuvres are mainly the surrounding traffic. The behaviour of the surrounding vehicles can be parameterised by available human car-following models, lane-changing models and decision-making models. In this project, to simplify the problems, these models are not used to make predictions.

The dynamic obstacles on the highways are the moving vehicles in the lanes. The behaviour of obstacles moving on the highway is not as complex as behaviour of obstacles moving in the urban conditions. The main challenge is the speeds of these obstacles moving on the highway are much faster than the speeds of the obstacles moving in the urban conditions.

The onboard sensors collect the information of the dynamic obstacles. The sensor system generates a real-time map according to the information given by these sensors. We are given

a list of dynamic obstacles by the real-time map, where each obstacle has a shape, a path and a velocity in the configuration space. Based on the location information of the moving obstacles in the configuration space at a specific time step, we can plan a feasible trajectory for the truck during this time step. In a configuration space, the states of the moving obstacles can be described by:

$$x_{ob} = [X, Y, \theta, V, G_a, G_f, N]^T$$

where (X, Y, θ) is the coordinates and the orientation angle in the inertial frame, V is the speed, G_a is the distance to the ahead obstacle, G_f is the distance to the follower obstacle, and the N is the information of the lane where the obstacle is located in.

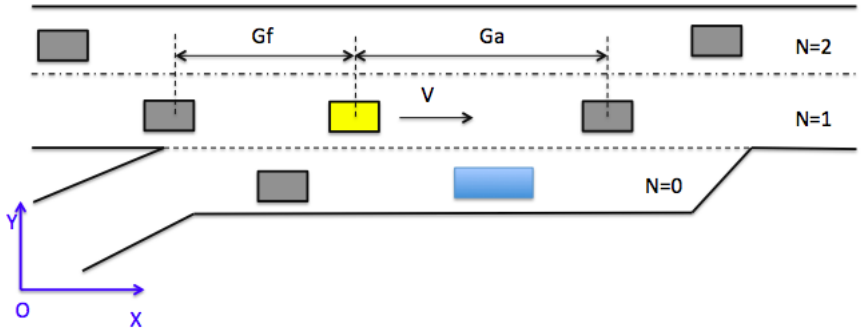


Figure 2-8: The geometric description of a dynamic obstacle in the highway. The yellow block is a dynamic obstacle, the blue block is the truck intended to merge into the highway, and the grey blocks are the other dynamic obstacles on the highway.

In a dynamic environment, the states of the moving obstacles are updated at every time step Δt. The states of these moving obstacles at time $(t + \Delta t)$ can be represented by the states of them at time t in Equation 2-15, where f is dynamic function that is related to the speeds of the obstacles. In this work, the function f is the motion equations of the obstacles in the configuration space.

$$x_{ob}(t + \Delta t) = x_{ob}(t) + f(V(t), \Delta t) \tag{2-15}$$

To simplify the modelling problem of the moving obstacles, we assume that when the truck starts to merge into the highway, no other vehicle intends to merge into the chosen gap. That means that we do not need to consider the effects of Y, θ, and N, since no lane-changing manoeuvre occurs between the putative leader vehicle (PL) and putative follower vehicle (PF) when the truck changes the lane. After the gap is chosen, the PL vehicle and PF vehicle are determined. Then we consider the relative distance between the PL vehicle and PF vehicle instead of G_a and G_f. The states of the obstacles can be described by Equation 2-16 in the configuration space at time $t+\Delta t$. We do not denote the distance between the PL vehicle and PF vehicle here, because when the dynamics of the PL vehicle and PF vehicle are determined, the relative distance between them are also determined.

$$\begin{bmatrix} X(t+\Delta t) \\ Y(t+\Delta t) \\ \theta(t+\Delta t) \\ V(t+\Delta t) \end{bmatrix} = \begin{bmatrix} X(t) \\ Y(t) \\ \theta(t) \\ V(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{2}a\Delta t^2 + V(t)\Delta t \\ 0 \\ 0 \\ a\Delta t \end{bmatrix} \tag{2-16}$$

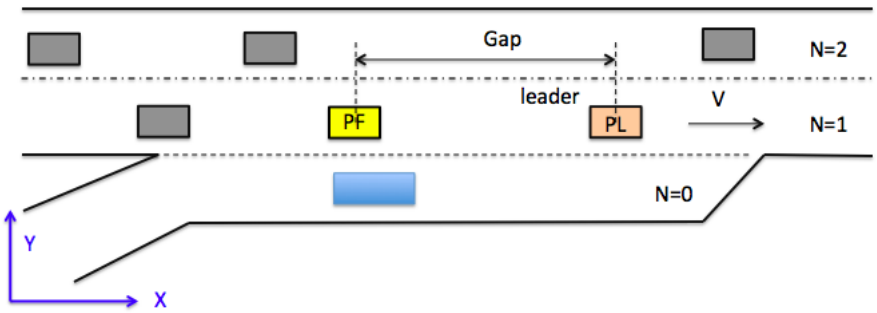


Figure 2-9: The geometric description of Staggered car-following behaviour on the highway. The yellow block is the putative follower vehicle, the blue block is the truck intending to merge into the highway, and the orange block is the putative leader vehicle in the target lane.

2-4 Summary

This chapter is mainly focused on the modelling problem in this project. We mainly discuss the modelling of the truck and the modelling of the obstacles in the traffic. In the first part, we represent the modelling of the truck. In the second part, we show the modelling of the obstacles.

The modelling of the truck plays a significant role of the motion-planning problem. In this project, we use a dynamic model including non-linear tyre properties with limits on longitudinal acceleration and steering angle, and with first order driveline and steering dynamics. Most of the kinematic models do not take the skid of the vehicle into consideration. They are restrictive and do not allow considering neither skidding when turning at high speed nor slip angle. Compared with kinematic models, the dynamic model has more complexity, and is hard to solve. To reduce the effects of the skid, we introduce a term G_{ss} [14]. The term G_{ss} is a static gain of the yaw rate $\dot{\theta}$, and is determined by the characteristic velocity V_{char} [15]. This term can reduce the influence caused by the skid and slip. Besides, we introduce a first-order lag term to model the delay caused by the actuators.

The obstacles in the highway can be classified into the static obstacles and dynamic obstacles. The states of the static obstacles do not change with time. We model the static obstacles in the configuration space by using the largest projected area of them in the $X-Y$ plane. In contrast to the static obstacles, the states of the dynamic obstacles change with time. To simplify the problem, we assume that no other lane-changing manoeuvres occur when the PL vehicle and PF vehicle are chosen. The dynamics of PL vehicle and PF vehicle can be described by their accelerations and speeds.

The truck model is used in the algorithm to ensure that the planned trajectory satisfies all the constraints of the truck. The obstacles model is mainly used in the obstacle avoidance algorithm. In the next chapter, we would like to introduce the main algorithms to solve the trajectory generation problem of the truck for merging manoeuvres on the highway.

Chapter 3

RRT Algorithm for Trajectory Generation

The previous chapter has shown that the truck model is a highly non-linear system with a set of linear and nonlinear constraints. The trajectory generated by the basic RRT algorithm cannot satisfy all the constraints of the truck. Actually, the trajectory generation problem of the truck belongs to the type of kinodynamic motion-planning problems whose general goal is to find a motion that goes from a start state to a goal state while satisfying all constraints of a nonlinear system. The RRT algorithm is one of the methods for the kinodynamic motion-planning problem. However, it cannot work well for this problem without corporation with other algorithms. In this chapter, we extend the RRT algorithm to cope with the trajectory generation problem of the truck.

3-1 The Basic RRT Algorithm

The Rapidly-exploring Random Tree (RRT) algorithm is an algorithm designed to efficiently search non-convex, high-dimensional spaces by randomly building a space-filling tree [17]. The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unsearched areas of the problem. Steven M. LaValle and James J. Kuffner Jr developed the RRT algorithm. The RRT algorithm handles problems easily with obstacles and differential constraints, and has been widely used in autonomous robotic path planning [17].

The basic RRT algorithm does not take the dynamics of the mechanical system into consideration. It spreads with a tree structure originating from initial state to a goal state in a configuration space X. The basic RRT algorithm first samples a node from the configuration space, and then the basic RRT algorithm finds a nearest node from the new sampled node in the tree; after this, the nearest node is driven to the new sampled node within a maximum

step length; at last, the collision-test function checks for any collisions between the new node and the infeasible space. If the new node lies in the collision-free space, it will be inserted into the tree. Every node in the tree has only one parent node. After reaching the maximum iteration number, it tracks back from a goal state to the initial state by calculating the lowest cost. The basic RRT algorithm includes five primitive procedures [10]:

Sampling: The samples are considered to be drawn from a uniform distribution, even though results extend naturally to any absolutely continuous distribution with density bounded away from zero on the configuration space X.

Nearest Neighbor: The graph $Tree = (V, E)$ is known and $V \subset X$. Given a point $x \in X$, the function $Nearest : (Tree, x) \mapsto v \in V$ returns the vertex in V which is closest to the point x according to a given distance function. The Euclidean distance is a frequently used method for determining the distance function.

Near Vertices: Assume a positive number $r \in R > 0$, the function $Near{:}(Tree, x, r) \mapsto V' \subseteq V$ returns the vertices in V which are contained in a ball of radius centered at point x.

Steering: Assume two points $x_1, x_2 \in X$, the function $Steer : (x_1, x_2) \mapsto z$ a point $z \in X$ such that z is closer to x_2 than x_1.

Collision Test: Assume two points $x_1, x_2 \in X$, the collision-detection function returns *True* if the line segment between x_1 and x_2 lies in the feasible space and *False* otherwise.

Considering a square configuration space, the frames in Figure 3-1 show the spreading of a RRT algorithm. The Basic RRT algorithm is represented by Algorithm 1. It takes the initial state as the root node of the spreading tree. Then randomly sample a node x_{rand} in the configuration space. The $Nearest(Tree, x_{rand})$ function tries to find the node $x_{nearest}$ that is closest to the random node x_{rand} in the tree. The $Steer(x_{nearest}, x_{rand})$ function connects the the $x_{nearest}$ and x_{rand} directly and marks x_{rand} as x_{new}, if the distance between these two nodes is smaller than a given distance r_d. Otherwise, a new state at the maximum distance from the tree along the line to the random sample is used instead of the random sample itself. The $Insert(x_{new})$ function inserts the new node x_{new} into the tree if x_{new} is a collision-free node. The algorithm stops if the maximum number of iterations is reached.

The RRT algorithm is a flexible method to solve the trajectory generation problem. The incremental nature of the algorithms lends itself easily to real-time, online implementation, while retaining probabilistic completeness guarantees [18]. The probabilistic completeness [19] is a weaker notion of completeness. A system is probabilistic completeness if the probability of returning a correct answer goes to 1 while the running time grows. Besides, the RRT algorithm is applicable to very general dynamical models. Additionally, the RRT algorithm does not require the explicit enumeration of constraints, but allow trajectory-wise checking of possibly very complex constraints.

However, there are some drawbacks for the RRT algorithm for the autonomous road vehicle motion-planning problems. The literature survey has pointed out the drawbacks: 1. RRT algorithm can easily get trapped when the tree spreads incorrectly. 2. One of the

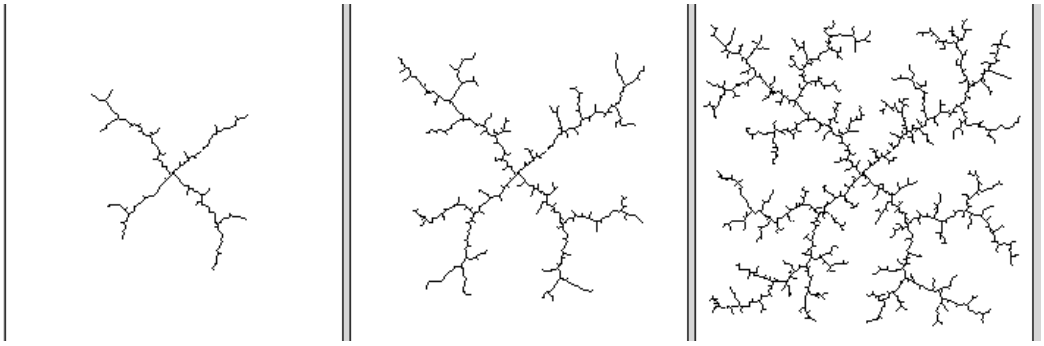


Figure 3-1: Construction of an RRT for $X = [0, 100] \times [0, 100]$, $\triangle x = 1$, and $x_{init} = [50, 50]$, from http://msl.cs.uiuc.edu/rrt/about.html

Algorithm 1 <Basic RRT >

Input: Initial configuration state x_{init}, maximum number of iterations n, sampling space ω
Output: RRT graph T
$T.init(x_{init})$
for i=1,...,n **do**
  $x_{rand} \leftarrow Sample_i(\omega)$
  $x_{nearest} \leftarrow Nearest(T, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $Collisiontest(x_{nearest}, x_{new}) = TRUE$ **then**
    $T(x_i) \leftarrow Insert(x_{new})$
  end if
end for
return T

stop conditions of such algorithms is often that the tree enters a certain goal region. There will be a risk, if the vehicle cannot reach the desired goal state. 3. The efficiency of RRT algorithm depends mainly on whether the sampling domain well adapts to the problem or not.

3-2 Kinodynamic Motion Planning

The general kinodynamic planning problem is to find a motion that goes from a start state to a goal state while satisfying all constraints of a nonlinear system. The kinodynamic planning problem usually combines a motion subject to simultaneous kinematic constraints, such as dynamics constraints, avoiding obstacles, and bounds on velocity, acceleration and force. Its solution is a mapping from time to generalised forces or accelerations. The resulting motion is governed by a dynamics equation [17].

The main procedures to build the roadmap (tree) with regard to kinodynamic path planning problem are:

1. Randomly choose an existing node
2. Randomly select the input vector u to the prediction system
3. Select integration time interval $\triangle T \in [0, t_{max}]$
4. Integrate equations of motion from an existing node with respect to u for time t
5. Check collision
   (a) If no collision occurs, store the control input with the new edge and add the new node to the roadmap
   (b) If collision occurs, ignore this node and repeat the former procedures

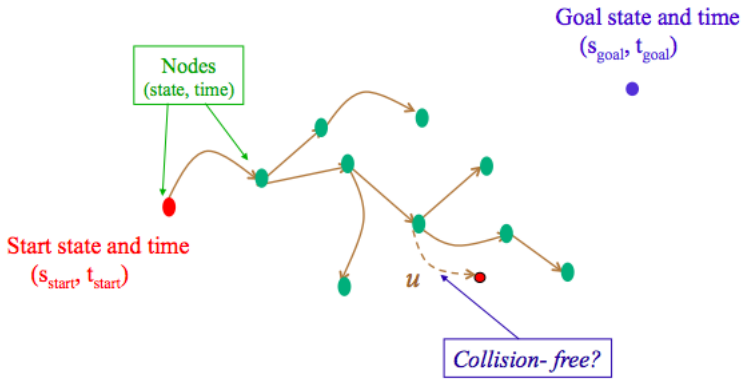


Figure 3-2: An example of building a roadmap between the start state and a given goal state. The roadmap extends by sampling existing nodes and sampling the input vector u

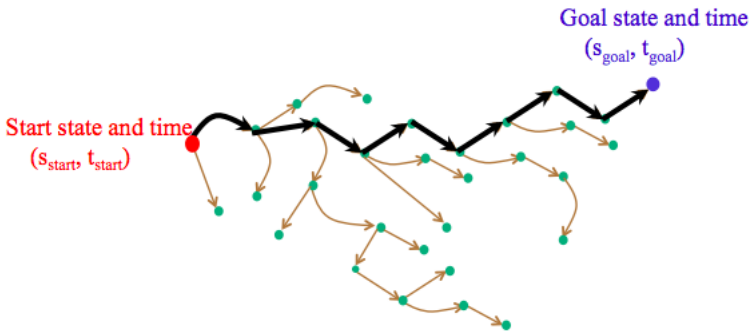


Figure 3-3: An example of a finding trajectory. After the goal state is reached, the algorithm will backtrack the roadmap to find the best trajectory

The basic RRT algorithm can find a path quickly; however, it may fail to work for kinodynamic motion planning that is a set of problems for which velocity, acceleration, and force/torque bounds must be satisfied, together with kinematic constraints such as avoiding obstacles [20]. Most of the systems with nonholonomic or holonomic constraints cannot follow the trajectory generated by the basic RRT algorithm. For solving this problem, a kinodynamic RRT algorithm is proposed to plan a trajectory for nonlinear systems with constraints.

The kinodynamic RRT algorithm differs from the basic RRT algorithm in several ways. The largest difference between the basic RRT algorithm and the kinodynamic RRT algorithm is whether the algorithm takes the system dynamics into consideration. The kinodynamic RRT algorithm considers the system dynamics when planning the motion, whereas the basic RRT algorithm only focuses on the spread of the tree.

3-3 Strategies for the RRT Algorithm

The RRT algorithm is a flexible and fast method for solving the trajectory generation problem. However, it has some disadvantages that impose negative effects on solving the motion-planning problem. The previous section has shown that the RRT algorithm has some drawbacks to solve the trajectory generation. In this section, we propose some useful strategies to improve the performance of the RRT algorithm.

3-3-1 Sampling Strategy

The RRT algorithm does not work efficiency for all the situations. When the configuration space contains a large quantity of points, the algorithm spends a lot of effort for the algorithm to achieve the goal. As the literature study mentioned, the RRT algorithm has the advantages of searching all the points of the configuration space in the same probability, and back tracking the nodes in the tree from a goal state to the start state quickly. However, in a structured

environment, a large number of samples are wasted due to the numerous constraints. To improve the efficiency, a bias can be added when sampling new points in the configuration space.

In the path-planning problem, we always want the tree to converge to the goal region as soon as possible. To address this, we can improve the sample probability of the nodes around the goal point by using the Gaussian distribution. In probability theory, the Gaussian distribution is a very common continuous probability distribution. The probability density of the Gaussian distribution is illustrated in Equation 3-1, where μ is the mean or expectation of the distribution, and the σ is its standard deviation with its variance σ^2.

$$f(x|\mu,\sigma^2) = \frac{1}{\sigma\sqrt{(2\pi)}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3-1}$$

The Gaussian distribution is a version of the standard normal distribution whose domain has been stretched by a factor σ (the standard deviation) and then translated by μ (the mean value). Based on the $3-\sigma$ rule, about 68% of values drawn from a normal distribution are within one standard deviation σ away from the mean; about 95% of the values lie within two standard deviations; and about 99.7 % are within three standard deviations. The goal point has the highest probability with probability density $\frac{1}{\sigma\sqrt{(2\pi)}}$ at $x = \mu$. We can find the distribution by adjusting the value of the mean μ and the deviation σ (see Figure 3-4).

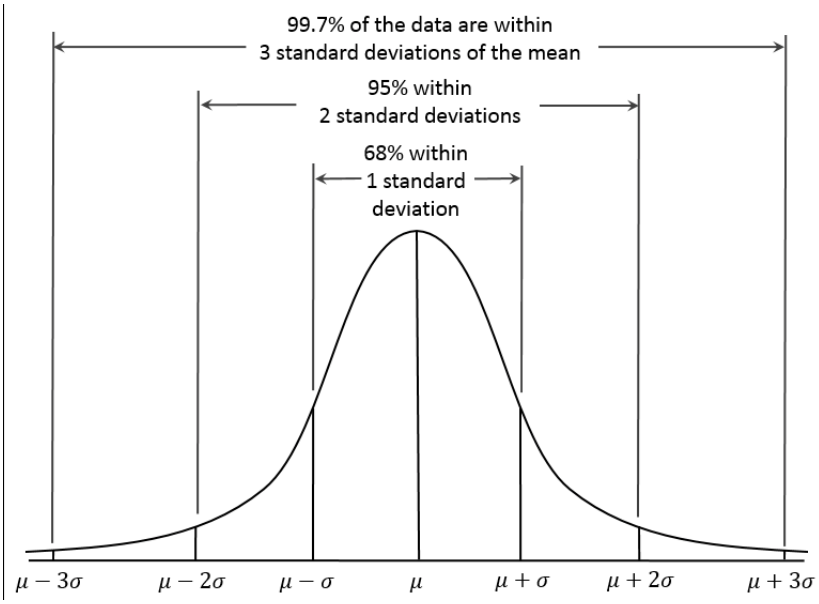


Figure 3-4: The Gaussian distribution. The values less than one standard deviation away from the mean account for 68.27% of the set of the entire points in the configuration space; while two standard deviations from the mean account for 95.45%; and three standard deviations account for 99.73%

To improve the sample probability of the points near the goal point, we can use the Gaussian distribution to narrow the sampling space of the RRT algorithm. The position of a vehicle can be described by points (X, Y, θ) in the inertia coordinate. Each sample (X_s, Y_s, θ_s) in the configuration space is generated with respect to a reference position and heading (X_0, Y_0, θ_0).

Moreover, the sample point can be mathematically described by Equation 3-2.

$$\begin{bmatrix} X_s \\ Y_s \\ \theta_s \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \\ \theta_0 \end{bmatrix} - r \begin{bmatrix} cos(\theta) \\ sin(\theta) \\ \sigma_\theta n_\sigma \end{bmatrix} \tag{3-2}$$

where $r = \sigma_r |n_r| + r_0$, n_r and n_σ are random variables with standard Gaussian distributions, σ_r and σ_θ are the standard deviation in the radial direction and circumferential direction, (X_0, Y_0) is the reference position, in this problem, it is the position of the goal, and r_0 is an offset with respect to the reference position. r_0 is used to prevent the value of r being equal to zero when the random number n_r is zero.

3-3-2 Node Connection Strategy

After sampling a new node in the configuration space, the next stage is to choose node to be added to the tree. If the nodes cannot be connected correctly, the algorithm will be trapped and fail to find a feasible path. Usually, the RRT algorithm attempts to connect the sample to the closest node in the tree, since shorter paths have a lower probability of collision. Thus, the RRT algorithm can quickly cover the free space without wasting many samples. There are several ways to evaluate the closeness. The simplest one is to use the two-norm distance to find the closest node in the tree.

The kinodynamic path-planning algorithm samples both the configuration space and the input space. Usually, a new node is inserted into the tree immediately if it is feasible, and only one trial is made from every closest node. If the trial returns an infeasible node, a new sample from the configuration space will be generated by the sample function. Here, we propose to make several trials with one sample from the configuration space, and several new nodes are generated at the same time with different samples from the input space, as Figure 3-5 shows. Too many trials slow down the algorithm, whereas too few trials make no sense. In this work, we make five trials with each sample. In order to make the algorithm converge to the goal quickly, the new node that is closest to the goal point will be selected to insert into the tree.

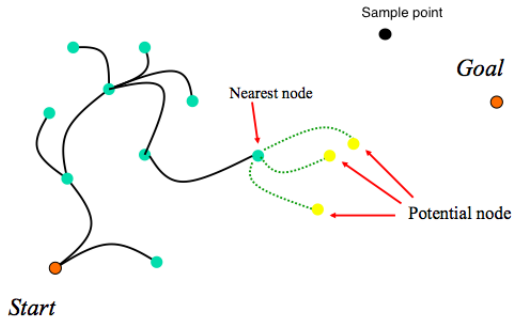


Figure 3-5: An example of the node selection in real-time planning. The yellow nodes are the potential nodes to insert into the tree. The dotted line is the potential path.

This strategy enlarges the number of samples in the tree, and reduces the sampling times of the algorithm. Usually, one sample from the configuration space generates no more than one feasible node to the tree. But with this strategy, more than one feasible node can be inserted into the tree with one sample from configuration space. More nodes in the tree means more nodes that can be chosen to construct the trajectory, and a better trajectory can be obtained.

Before inserted the node into the tree, we only check the collision as usual. But for a system with constraints, the two criteria are not enough. Since the vehicle has its own limitations, the collision-free nodes are not all feasible for the vehicle dynamics. That is, the inputs that drive the vehicle moving from the parent node to the child node in the tree are beyond the limitations of vehicle. It is impossible for the vehicle to complete this action. To generate a collision-free and vehicle-feasible trajectory, the collision-free nodes that do not satisfy the vehicle dynamic limitations should be ignored as well.

3-3-3 Algorithm Stop Strategy

The propagation algorithm will stop if some criteria are satisfied. The most frequently used criterion is the criterion of the goal region in the configuration space. Once a feasible node is found inside the goal region, the RRT algorithm stops.

The goal region imposes obvious impacts on the efficiency of the algorithm. If the region is too narrow, the algorithm may find no feasible trajectory. If it is too large, the algorithm may stop too early. When a node appears near the goal state in a definite region, the RRT algorithm will stop. The region of the goal is important for the RRT algorithm to find a feasible path, especially for real-time planning. Usually, we use a sphere region centred by the goal with a radius R, but in the highway this method may lead to an inaccurate result if the radius R is too large. In addition, we do not want the truck to deviate from the lane too much when merging into the highway. Therefore, a polygon region can be used here.

The goal point centres the polygon goal region. Actually, the polygon goal region is a rectangle with small width W_{rec} and large length L_{rec}. Since the width of a truck is $W = 2.5\ m$ and the road is only $W = 3.5\ m$ in width, the width of the polygon cannot exceed $W_{rec} = 0.5\ m$ for ensuring the safety. As for the length of the rectangle, we can choose a relative large value for it, since the value of the gap's length is much larger than the value of the road's width.

In a configuration space, we use not only the coordinates pair (X, Y) of the anchor point to describe the location of the vehicle, but also use the orientation angle θ to describe the posture of the vehicle. Usually, the goal region of the basic RRT algorithm is a stop criterion only considering the coordinates pair (X, Y), not including the orientation angle θ for the purpose of speeding the algorithm. However, for the merging manoeuvres, the orientation angle θ of the truck also makes sense for the trajectory. If the value of the orientation angle θ does not satisfy a certain criterion, a collision will also occur in the reality even if the criterion of the coordinate location is fulfilled. Therefore, an extra stop criterion with respect to θ is needed. The stop criterion for the orientation θ depends on the scenario and the final state.

More criteria mean more efforts to achieve the goal. However, it could obtain accurate results. If we use many criteria for all the propagation process, the time to achieve the goal may become huge. Therefore, we propose the combined stop criteria here. The combined stop criteria use different stop criteria when the goal point is close and when the goal point is far away. The combined stop criteria are actually the balance of the accurate solution and small computational cost.

For the merging manoeuvres of the truck, we would like the truck enter into the target lane with small orientation deviation first; then adjust its orientation angle to an acceptable value. To balance the accuracy and the cost, the criteria with wide range value of XY coordinates and θ are used before the truck enters into the target lane, and the criteria with narrow range value of then are used after the truck enters into the lane.

3-4 Offline Trajectory Generation

The offline trajectory generator is to generate a trajectory when the environment information is not updated by time. It is the basis for realising the real-time implementation. There are two different ways for solving the off-line trajectory generation problem: the open-loop RRT algorithm, and the closed-loop RRT algorithm. In this section, we would like to introduce the offline RRT algorithm over open-loop dynamics and over closed-loop dynamics.

3-4-1 Offline Planning over Open-Loop Dynamics

The so-called open loop refers to the dynamic system that does not take the controller into consideration. Every new node inserted into the tree is obtained by solving the dynamic equations. The configuration of the planner with open-loop prediction system is illustrated in Figure 3-6.

The RRT algorithm samples the control input signals that can drive the vehicle moves towards the goal region. The input signals can be acceleration, steering rate, gas, velocity, steering angle, and so on. The algorithm samples a sequence of input signals u at every time step. This sequence of input signals u drives the vehicle moves from the nearest node to a new node during this time step. Then the new node is recorded at the end of this time step if this new node is collision-free. Algorithm 2 is the pseudo code of the open-loop RRT trajectory planning algorithm.

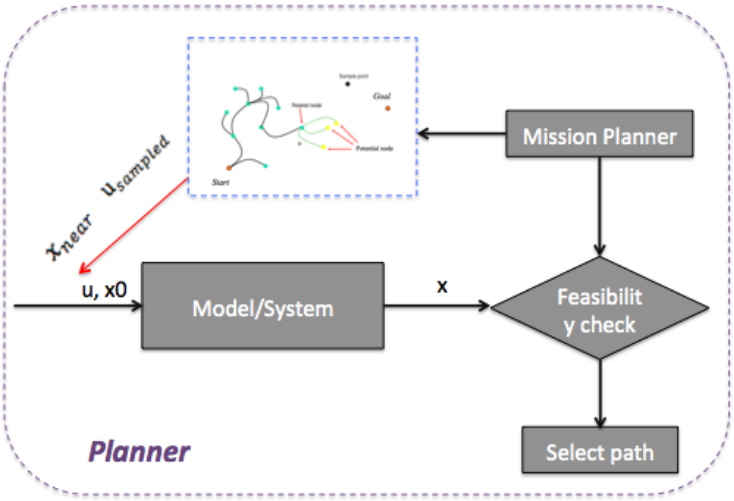


Figure 3-6: The configuration of the planner with open-loop prediction system. Given a sequence of inputs signals u, and then collect the output x to the tree when x is in the feasible space.

Algorithm 2 <Open-loop RRT Algorithm>

Input: Tree $\mathfrak{Im}$, Goal region $\mho$, Timestep Δt, i=0, Initial state x_{init}, $x_0 = x_{init}$
while $x_i \notin \mho$ **do**
  $i + 1 \leftarrow i$
  $x_{sample} \leftarrow RAND\_CONF()$
  $x_{nearest} \leftarrow NEAREST\_NODE(x_{sample}, \mathfrak{Im})$
  $u_{sample} \leftarrow RAND\_INPUT()$
  $x_{new} \leftarrow SIMULATION\_SYSTEM(x_{nearest}, u_{sample}, \Delta t)$
  if $COLLISION(x_{new}) = FALSE$ **then**
    $\mathfrak{Im} \leftarrow INSERT\_NODE(x_{new})$
    $x_i \leftarrow x_{new}$
  end if
end while

Algorithm 2 samples both the configuration space and the input space. The $RAND\_CONF()$ function samples a node x_{sample} in the configuration space. The $NEAREST\_NODE()$ function finds out the nearest node $x_{nearest}$ to the x_{sample} in the tree. Then the function $RAND\_INPUT()$ samples a set of input signals u_{sample} to forward the vehicle from $x_{nearest}$. The new node x_{new} is obtained by function $SIMULATION\_SYSTEM()$. After this, the function $COLLISION()$ checks the feasibility of the new node x_{new}. If it is feasible, the function $INSERT\_NODE()$ adds the new node x_{new} into the tree. Otherwise, the new node is ignored. The algorithm stops when there is a node in the tree appearing in the goal region.

A frequently used method to solve the system's motion equations in path planning problems is the 4th-order Runge-Kutta integration method [21]. The 4th-order Runge-Kutta method is based on a fourth order Taylor's approximation. It predicts the next value by the present

value plus the weighted average of four increments, where each increment is the product of the size of the interval, and an estimated slope specified by function on the right-hand side of the differential equation. Assumed the nonlinear system can be described by Equation 3-3, where y is an unknown function of time t that we need to approximate, u is the input of this function, and at $t = t_0$, we already have $y = y_0$.

$$\dot{y} = f(y, u, t), \qquad y(t_0) = y_0 \tag{3-3}$$

We denote $y_{n+1} = y(u, t + \Delta t)$ and $y_n = y(u, t)$. By choosing a suitable positive time step-size Δt, we can make an approximation to y_{n+1} by using y_n and weighted increments. The approximation algorithm is represented in Equation 3-4.

$$\begin{aligned} y_{n+1} &\approx y_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ t_{n+1} &= t_n + \Delta t \end{aligned} \tag{3-4}$$

where

$$\begin{aligned} k_1 &= f(y_n, u_n, t_n), \\ k_2 &= f(y_n + \frac{k_1}{2}, u_n, t_n + \frac{\Delta t}{2}), \\ k_3 &= (y_n + \frac{k_2}{2}, u_n, t_n + \frac{\Delta t}{2}), \\ k_4 &= (y_n + k_3, u_n, t_n + \Delta t) \end{aligned} \tag{3-5}$$

We sample the points in the configuration space C to extend the tree, and sample the input vector $u \in U$ to the generated new node of the trajectory. It is possible to consider the sampling strategy with these two sampling processes at the same time. It is also possible to add a bias only to one of the sampling process. For the kinematic model described by Equation 2-8, we have the elements in the configuration space C and input space U:

$$\begin{aligned} [X, Y, \theta]^T &\in C \\ [a, r]^T &\in U \end{aligned}$$

3-4-2 Offline Planning over Closed-Loop Dynamics

In the previous section, we solve the nonlinear system with a mathematic method. The model in the planner is without controllers. In this subsection, we use the closed-loop RRT algorithm to plan a trajectory for the truck.

There are obvious advantages of the closed-loop RRT algorithm compared with the open-loop RRT algorithm. The benefits are extremely obvious when the vehicle model is not stable, and the model has noise and system uncertainty. The use of a stabilising controller provides a smaller prediction error because it reduces the effect of any modelling errors in the vehicle dynamics on the prediction accuracy, and also rejects disturbances that act on the actual vehicle [14].

The closed-loop RRT algorithm differs from the open-loop RRT algorithm. The open-loop

RRT algorithm samples the points in the configuration space as well as the inputs signal in the input space. The planner with open-loop RRT algorithm samples inputs u from the input space U; and then send the sampled input signals to the system to obtain the new node added to the tree. The closed-loop RRT algorithm samples the configuration space only. Actually, the closed-loop RRT algorithm samples the signals to the controller. The planner with closed-loop RRT algorithm samples the reference segments r; and then send r to the closed-loop system. The Figure 3-7 illustrates the configuration of the planner with a closed prediction loop inside. Algorithm 3 shows the pseudo code of the closed-loop RRT algorithm.

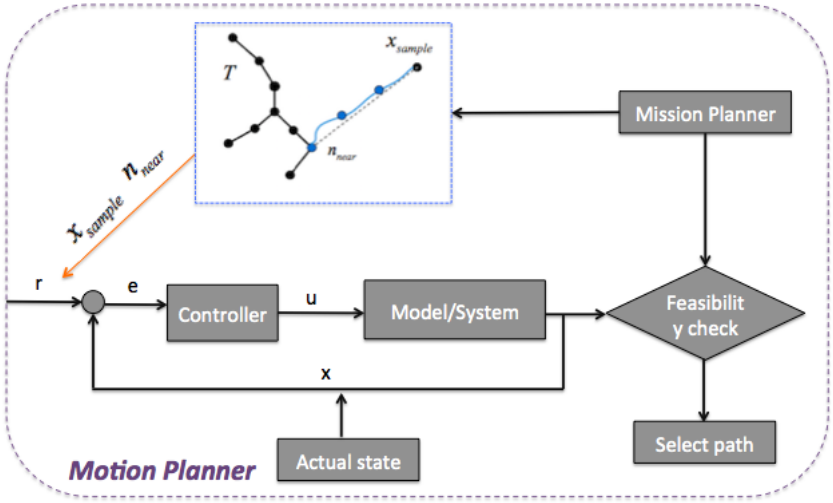


Figure 3-7: The configuration of the planner with a closed-loop system. Given a reference command r, the controller will generate commands u to system.

Algorithm 3 <Closed-loop RRT Algorithm>

Input: Tree $\mathfrak{Im}$, Goal region $\mho$, Timestep Δt, Initial x_{init}, $x_{init} = x_0$, i=0
while $x_i \notin \mho$ **do**
  $i + 1 \leftarrow i$
  $x_{sample} \leftarrow RAND\_CONF()$
  $x_{nearest} \leftarrow NEAREST\_NODE(x_{sample}, \mathfrak{Im})$
  $x_{new} \leftarrow STEER(x_{sample})$
  $x_{new} \leftarrow SOLVE\_SYSTEM(x_{nearest}, x_{new}, \Delta t)$
  if $COLLISION(x_{new}) = FALSE \&\& CONSTRAINT\_CHECK(x_{new}) = SATISFY$ **then**
    $\mathfrak{Im} \leftarrow INSERT\_NODE(x_{new})$
    $x_i \leftarrow x_{new}$
  end if
end while

Algorithm 3 is similar to Algorithm 2. The key difference is that Algorithm 3 does not sample the nodes in the system inputs space. As a planner, each node in the closed-loop RRT algo-

rithm stores the state of vehicle x and the controller input r. The input given to the controller is a series of straight lines called the reference path. The planner generates this reference path by connecting a sample to the controller input at a node in the tree. For each sample, a node is selected from the tree by using $NEAREST\_NODE(x_{sample}, \mathfrak{Im})$ in Algorithm 3. Then connecting the sample node and the selected node generates the reference path. By using this straight line as an input, the $SOLVE\_SYSTEM()$ in Algorithm 3 drives the state to a new state from the selected node. In Algorithm 3, the function $CONSTRAINT\_CHECK$ is used to select the nodes that do not satisfy the vehicle dynamic limitations. The algorithm is stopped when the tree contains a node in the goal region.

3-5 Real-Time Trajectory Generation

In the previous section, we introduce the RRT algorithm for a static environment. That is, the surrounding traffic is assumed not to move when the algorithm plans a trajectory. In this chapter, we will focus on the RRT algorithm works in a dynamic environment.

The dynamic environment means that there exist moving obstacles in the environment. In the highway, the road boundaries and the lanes never change with time; hence they are static, and not considered in the dynamic environment. Each dynamic obstacle has its own dynamics states in a configuration space.

Real-time trajectory generation generates a trajectory for the mechanics system in a dynamic environment. The RRT algorithm plans a trajectory for the system from the state $x(t)$ to the state $x(t + \Delta t)$. The dynamic environment will be updated every time step Δt; and then a replanning algorithm is executed to forward the plan. Figure 3-8 shows the flow chart of the real-time implementation.

In a dynamic and uncertain environment, the algorithm needs to be aware of the changes of the surrounding traffics in order to generate a safe trajectory. The real-time execution requires reusing the necessary information from the previous states. Algorithm 4 is the pseudo code of the real-time loop of the RRT algorithm.

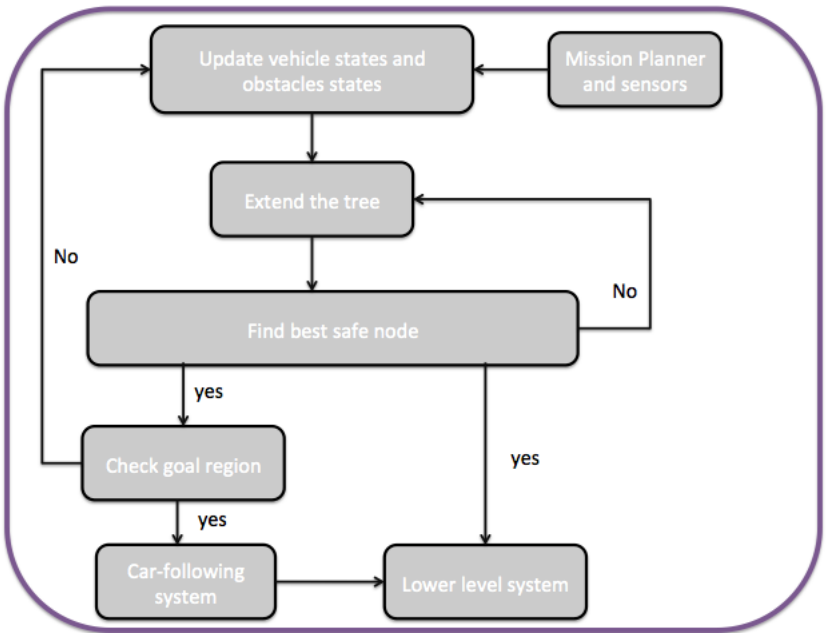


Figure 3-8: The flow chart of the implementation of the real-time RRT algorithm .

Algorithm 4 <Real-time RRT Algorithm>

Input: Tree $\mathfrak{Im}$, Goal region $\mho$, Time step Δt, Initial x_{init}, Initial environment state $\chi(t)$
while No node in the Goal region **do**
  Update the current vehicle states and the dynamic obstacles' states
  while No best node sequence can be found **do**
    Extend the tree until Δt is reached
    Choose the best safe node sequence $path_i$ in the tree
  end while
  Send the chosen node sequence $path_i$ to the system
  Empty the tree
end while

Algorithm 4 outlines the replanning procedure with the RRT algorithm while the lower level controller executes the plan. The planner sends the input to the lower level system at a fixed rate of every Δt seconds. The tree expansion continues until Δt is reached. The best trajectory is selected and the input is sent to the lower level system for execution. The expansion of the tree is resumed after updating the vehicle states and the environment.

Unlike in the urban driving situation, it is not safe to command an emergency braking manoeuvre for the highway driving situation, especially when the traffic flow is high. In order to avoid the emergency braking manoeuvre on the highway, the time step size Δt cannot be large. A larger Δt can predict a longer trajectory, however, it also increases the collision probabilities.

One way to implement the RRT-based planner is to build a new tree at every planning

cycle and discard the old tree. In this way, a path is selected for executing the lower level system without considering the current path being executed. Every planning cycle after a path is selected and sent to the lower level system, the planner will restart to plan a new motion from the original state. This means almost identical computations would have to be repeated. In a real-time application, this way to implement the planner is not efficient, and will waste a lot of computational resources. Additionally, if discarding all the information in the old tree, the trajectory generated by a new cycle may be independent on the trajectory generated by the old tree. This means the planner could switch between two different trajectories at every planning cycle, which may lead to planning a wavy trajectory for the merging manoeuvre in the dynamic environment.

The second way to implement the RRT-based planner is to build a new tree at every planning cycle based on the old tree. This way is also not an efficient way to implement the RRT-based planner in the real-time. One reason is that the tree will grow larger and larger, and requires a lot of space to store the nodes of the tree. Another reason is that it needs more effort to find the best nodes sequence when the tree contains too many nodes. Additionally, implementing the planner in this way is easy to plan a stack trajectory.

Another way to implement the RRT-based planner is to build a new tree based on the end node information of the trajectory generated by the old tree. In this way, after every planning cycle, the end of the generated trajectory is stored and used as the root of the new tree in the next planning cycle. But all the other children branches from the old tree are then deleted because these branches will never be executed. The new tree then grows in its planning cycle until Δt is reached. After the new trajectory is found, its end node will also be stored as the root of the tree in the following planning cycle, and other branches of the new tree will be deleted. Therefore, the trajectory executed by the lower level system is always continuous.

Both Algorithm 2 and Algorithm 3 can be used to extend the tree during the time step size Δt. If the open-loop RRT algorithm is used to extend the tree in a real-time implementation, the inputs to the vehicle are sent to the lower level system to execute the plan, whereas if the closed-loop RRT algorithm is chosen to extend the tree, the reference paths are sent to the controller of the lower level system to execute the plan.

3-6 Obstacle Avoidance

The RRT algorithm finds a path in the configuration space. The path must be checked before it is accepted as valid. There exist a lot of collision detection algorithms handling obstacles with different complexibilities. The obstacles with complex geometry need advanced detection algorithm, whereas advanced detection algorithms mean more computational costs. In this work, we consider the obstacles with regular shapes in two-dimensional space; hence, the simple intersections detection method is proposed.

Intersection detection is the problem of detecting whether two objects intersect or overlap in space [22]. Performing intersection tests carries out intersection detection. It is a problem that occurs in computer graphics in many forms, including: clipping, view-volume culling,

ray-tracing, picking and collision detection. Intersection detection is the heart of collision detection, and collision detection is the intersection detection with regard to a set of moving objects.

3-6-1 Bounding Volumes

Bounding volumes (BVs) are the basic and efficient technique to solve the intersection detection problem [23]. It is often useful to have a bounding volume enclosing a finite geometric object. Bounding volumes can significantly speed up software for ray tracing, collision avoidance, hidden object detection, etc. Before invoking the computationally expensive intersection or containment algorithms for a complicated object, a simple test with an uncomplicated bounding volumes can often exclude the possibility of intersection or containment, and no further wasteful computation is needed.

In order to use a bounding volume as a part of a collision detection algorithm, we must firstly fit a bounding volume to any given collection of primitives that capture the main shape information of the obstacles, and then determine whether two given bounding volumes overlap. In this projects, we regard the largest size of a vehicle as the primitives. There are many types of bounding volumes: sphere, oriented bounding box, axis-aligned bounding box, convex hull, ellipsoid, prism, cub, and many more. The most frequently used bounding volumes are sphere, axis-aligned bounding box and oriented bounding box [5].

There exist several types of bounding boxes as shown in Figure 3-9: the sphere [24], the axis-aligned bounding box (AABB) [25], the oriented bounding box (OBB) [26], the 6-DOP [27], and the convex hull [28]. Different types of bounding boxes have different complexities, costs and tightnesses of fit. A more complex bounding box means a better tightness of fit, but it costs more to test the overlap and to update. As the vehicle can be regarded as a regular shape and the efficiency is important in real-time implementation, too complex bounding boxes are not considered in this project.

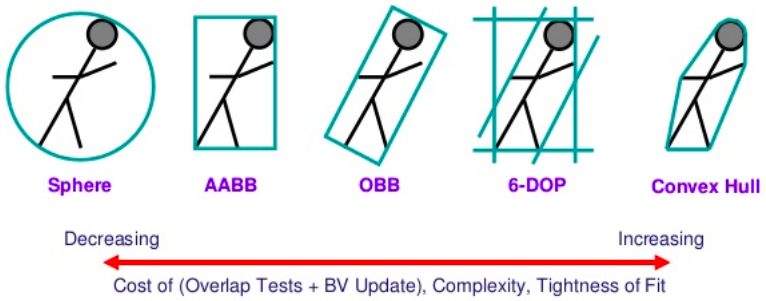


Figure 3-9: Different types of bounding boxes.

The axis-aligned bounding box (see in Figure **??**) is a box that defines a rectangular region whose edges are parallel to the coordinate axes, and is thus defined by its maximum and

minimum extending for all axes. In two-dimensional space, the bounding box is given by (x, y) coordinates that satisfy Equation 3-6, and specified by the extreme points (x_{min}, y_{min}) and (x_{max}, y_{max}), which are its bottom-left and top-right corners. Alternatively, it can be specified as a corner point (c_x, c_y) and edge lengths l_x, and l_y, as in Figure 3-7.

$$D = \{(x, y)|x_{min} \leq x \leq x_{max}, y_{min} \leq y \leq y_{max}\} \tag{3-6}$$

$$D = \{(x, y)|c_x \leq x \leq c_x + l_x, c_y \leq y \leq c_y + l_y\} \tag{3-7}$$

An oriented bounding box is a rectangular shape. It is similar to an axis-aligned bounding box except that it has arbitrary orientation. An oriented bounding box can be described by a centre point c, edge half-lengths r_1 and r_2, and two orientation unit vectors $\vec{v_1}$ and $\vec{v_2}$. The oriented bounding box is illustrated in Equation 3-8.

$$D = \{c + ar_1\vec{v_1} + br_2\vec{v_2}|a, b \in [-1, 1]\} \tag{3-8}$$

In addition to the axis-aligned bounding box and the oriented bounding box, the bounding sphere, represented in Equation 3-9, is another type of method to model a geometric object. It is the smallest circle or sphere containing the object. It is also called the "minimal spanning sphere" of the object. The bounding ball of a geometric object is unique, and is specified by a centre point $C = (c_x, c_y)$ and a radius r. It is easy to detect whether a point is inside a bounding sphere by checking that it is within distance r of the centre point C. Besides, it is also easy to detect whether two bounding spheres are disjoint by checking the distance of two centre points.

$$D = \{(x, y)|(x - c_x)^2 + (y - c_y)^2 < r^2\} \tag{3-9}$$

3-6-2 Intersection Detection

The intersection detection may report merely a complete description of the set intersection of the objects, possibly reporting whether two objects in a given pose are touching. If two objects are disjoint, the intersection detection algorithm will return false, otherwise, it will return true. We determine the intersection of two objects by checking whether there exists a joint point belonging to both of these objects. If so, the collision occurs, otherwise no collision occurs.

The frequently used test algorithms in geometric methods are sphere-sphere test algorithm, sphere-AABB test algorithm, sphere-OBB test algorithm, the OBB-OBB test algorithm, AABB-AABB test algorithm and OBB-AABB test algorithm [5]. These algorithms are used to detect the overlap of two objects. The objects can be modelled as one of these bounding volumes in the configuration space. If the test algorithm returns true, there exists overlap between these two objects. If not, the two objects are disjoint.

All of these collision test algorithms are based on a point-line detection method, a point-point detection method and a line-line detection method. These methods use the distance between these two elements to determine whether there exists an intersection. There are several varieties of distance measure. These are measures which yield a single real number

that describes the distance between two sets A and B. The simplest separation distance is used here. The separation distance is the length of the shortest line joining the sets. The form of this distance is illustrated in Equation 3-10.

$$dist(A, B) = \min_{a \in A} \min_{b \in B} |a - b| \tag{3-10}$$

The geometric elements are described in the inertial coordinate system. The equations to calculate the distance of two points, point-line and line-line in a coordinate system are Equation 3-11, Equation 3-12 and Equation 3-13,respectively.

$$dist(P_1, P_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3-11}$$

where P_1 and P_2 are two points with coordinates (x_1, y_1) and (x_2, y_2).

$$dist(L_{ab}, P_0) = \frac{|(y_b - y_a)x_0 - (x_b - x_a)y_0 + x_b y_a - y_b x_a|}{\sqrt{(y_b - y_a)^2 + (x_b - x_a)^2}} \tag{3-12}$$

where $P_a = (x_a, y_a)$ and $P_b = (x_b, y_b)$ are the two points in the line L_{ab}, and $P_0 = (x_0, y_0)$.

$$dist(L1, L2) = \frac{|(c - a)[(b - a) \times (d - c)]|}{|(b - a) \times (d - c)|} \tag{3-13}$$

where $L1 = a + (b - a)s$ and $L2 = c + (c - d)t$.

3-6-3 Collision Avoidance on the Highway

There exist static obstacles and dynamic obstacles on the highway when considering the collision avoidance problem. The dynamic obstacles are static at specific time steps in the simulation. Therefore, in a specific time step, the states of a dynamic obstacle do not change. We can check the intersection of the bounding volume of the vehicle and the bounding volume of the surrounding obstacles at a small time step to determine the collision.

Before applying the intersection detection algorithm to check the collision, bounding volumes are chosen for the vehicle and the surrounding obstacles. There are three types of bounding volumes fitting for truck since the shape of the truck can be looked as an oblong shape in 2D configuration space. The three bounding volumes are sphere as shown in Figure 3-10, OBB as shows in Figure 3-11 and AABB as shows in Figure 3-12.

As for a truck, a too loose bounding volume is not suitable due to its large length. The length between the front wheel axle and rear wheel axle is $l = 5\ m$, and the width of the truck is $w = 2.5\ m$. Actually, the length of the truck is larger than the length of the wheelbase, so when modelling the truck in a configuration space with bounding volumes, the length of the truck will be chosen as $L = 7\ m$. If the sphere is used to model the truck, the smallest radius of the sphere will as large as $r = 3.5\ m$. A lot of safe nodes will be ignored because the width of the road is only $w_r = 3.5\ m$. As a consequence, no feasible trajectory may be found by the algorithm.

In this work, we choose an OBB for the truck, since OBBs have better tightness of fit than

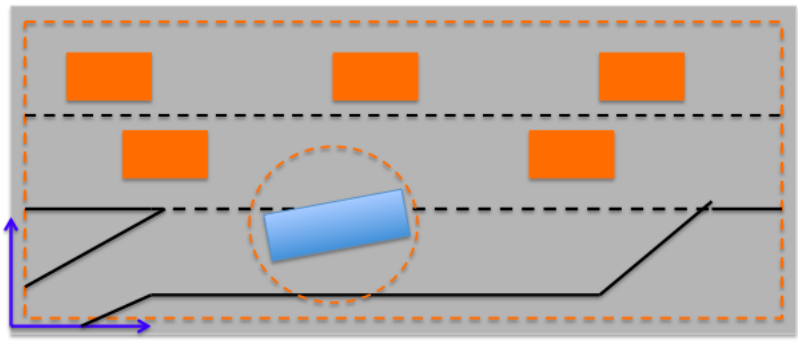


Figure 3-10: Modelling the truck as a sphere in a configuration space when checking the collision.

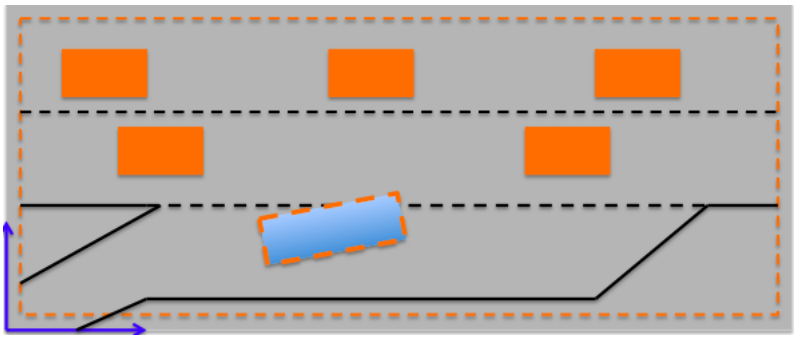


Figure 3-11: Modeling the truck as an OBB in a configuration space when checking the collision.

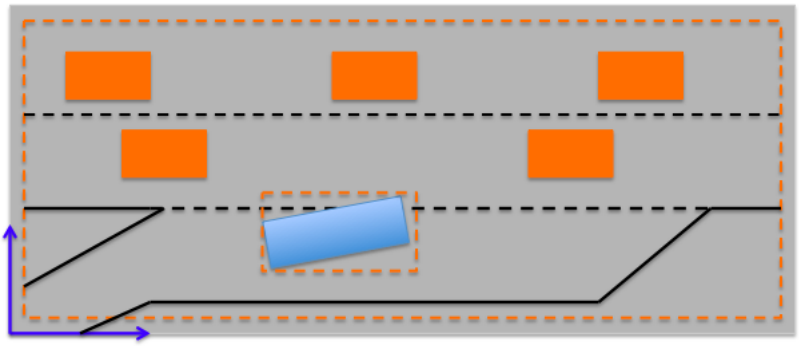


Figure 3-12: Modeling the truck as an AABB in a configuration space when checking the collision.

AABBs. After determining the bounding volume for the truck, the bounding volumes for the surrounding obstacles can be chosen. Different types of bounding volumes have different efficiency in overlap test with OBB. The [5] has made experiments on the time of overlap testes for different bounding volumes with OBBs. The results are listed in Table 3-1. The [5] pointed out that the sphere-OBB overlap test needs the lowest time, whereas the OBB-OBB overlap test costs more time than the other two types in overlap test.

Method used	Disjoint (μs)	Intersection (μs)
Sphere	0.97	0.92
AABB	1.36	1.38
OBB	5.09	7.44

Table 3-1: Timing of overlap testes for different bounding volumes. The AABB represents the axis-aligned bounding box, and the OBB represents oriented bounding box. These data are from [5].

There exist static obstacles and dynamic obstacles in the traffic. Although they are all regarded as static at a specific time step, we still consider these two types of obstacles separately here. The static obstacles are the road boundaries that are modelled as polygons based on its size in the configuration space. The algorithm will ignore every point of the truck located in this polygon. Although the sphere-OBB test shows highest efficiency, the AABB is a better choice for the surrounding vehicles, since we can use an uniform obstacle detection method for both static and dynamic obstacles. To conclude, AABB is chosen for the bounding volume of surrounding vehicles, and the road boundaries are modelled as polygons in the configuration space.

To improve the safety, tail spaces can be attached on the rear of vehicles in the traffic. This tail space, as shown in Figure 3-13, can be regarded as a buffer for avoiding the collision caused by the inertia forces and the response latencies of the vehicles. In this project, the bounding volume of the truck are 3 m in width and 7 m in length. The tail space of the truck is chosen for 2 m in length which is the stand-off safety distance of vehicles, and 3 m in width.

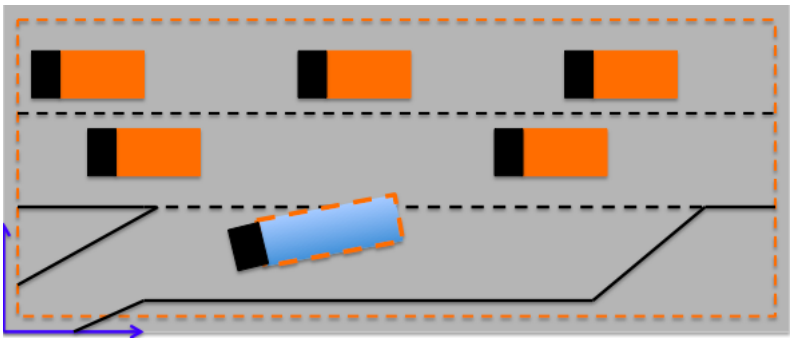


Figure 3-13: The tail space attached on the the rear of the vehicles, the black blocks are the tail spaces.

As we have already chosen the bounding volumes for all the objects of the highway in the configuration space, the shape of the bounding volumes can determine a suitable collision detection method. All the chosen bounding volumes are convex and regular. The simplest method to detect the collision of two convex bounding volumes is the line-line intersection method. If the lines constructed by connecting any two of the close vertexes of each bounding volume have no intersection with other bounding volumes, then no collision occurs.

The vertexes of the truck can be represented by the coordinates of its centre of gravity (CoG) in the configuration space. We have used OBB for the truck, therefore, there are four vertexes of the trucks as illustrated in Figure 3-14. Assumed the position of CoG is described by C=(X_c, Y_c) and the orientation angle is denoted as θ_c in the configuration space, we can determine the coordinates of these vertexes by Equation 3-14.

$$\{(X,Y)|(X_c + aL_r cos(\theta_c) - b\frac{W}{2}sin(\theta_c), Y_c + aL_r sin(\theta_c) + b\frac{W}{2}cos(\theta_c))|a,b \in [-1,1]\} \quad (3\text{-}14)$$

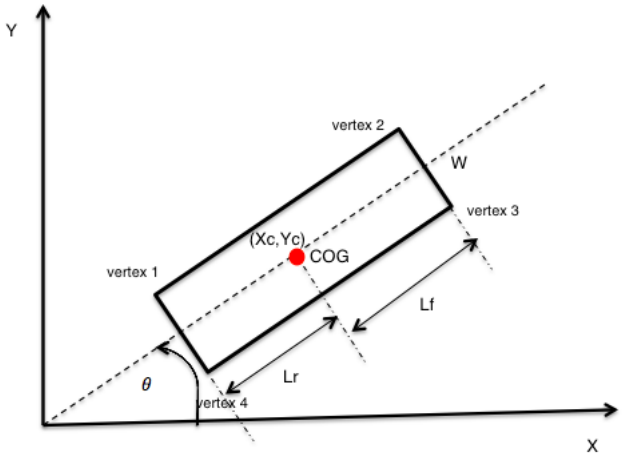


Figure 3-14: The posture of the truck in the configuration space. The (X_c, Y_c) is the coordinate of the CoG in the inertial coordinate system, the θ_c is the heading angle when the truck's CoG located in (X_c, Y_c).

When checking whether the new node that is obtained by solving the vehicle dynamic equations is feasible or not, we can repeat the line-line intersection detection algorithm with the line segments constructed by these four vertexes and the obstacles.

3-7 Summary

In this chapter, we address the main solution for the trajectory generation problem. At the first part, we introduce the RRT algorithm. Then we depict the RRT algorithm used for solving the kinodynamic path-planning problem. After this, some strategies for improving the performance of the algorithm are proposed. In the middle of this chapter, we present the

solutions for the off-line trajectory generation and for the on-line trajectory generation. At the end of this chapter, the solution for the obstacle avoidance is illustrated.

The RRT algorithm has five primitive procedures: Sampling, Nearest Neighbour, Near Vertices, Steering, and Collision Test. The basic RRT algorithm does not take the dynamics of the system into consideration. The path planned by the basic RRT algorithm is usually impossible for a real vehicle to follow, since the vehicle has its own dynamic limitations. The main difference between the basic RRT algorithm and kinodynamic RRT algorithm is whether the node that will be chosen to insert into the tree is obtained by solving the dynamic equations. The kinodynamic RRT algorithm is the basic solution for the trajectory generation problem. In order to improve the performance of the algorithm, some strategies are proposed then.

We discuss about the strategies used with the algorithm to improve the efficiency. The sampling strategy is to add a bias to the algorithm so that the nodes near to the goal region have larger probabilities to be sampled. The node connection strategy is to add more choices when selecting a node inserting to the tree. The stop strategy is to combine different criteria to make a trade-off between the computational cost and the accuracy of the results.

The main part of this chapter is the off-line algorithm and the online replanning algorithm for trajectory generation. There exists two approaches for the off-line planning problem. One approach is the off-line path planning over the open-loop dynamics, and the other is the off-line path planning over the closed-loop dynamics. The main difference between these two approaches is the dynamic system used for predication. With regard to the replanning algorithm, we introduce three different ways to extend the tree in real-time implementation. The way uses the last information of the former planning cycle, which can avoid to do the repeated work and to construct a huge tree, is a good choice for extending the tree in the real-time implementation.

The collision detection is an essential part with regard to the trajectory generation. To be compatible with the RRT algorithm, we use bounding volumes to model the truck, as well as obstacles, in the configuration space. In this work, we choose the OBB for modelling the shape of the truck in merging maneuvers, while the AABBs are chosen for modelling the vehicles surrounded the truck. No matter which kind of bounding volumes are chosen for the objects on the highway, the basic collision detection method is to check the intersection between points and lines. As we do not choose the sphere as bounding volume, only the line-line intersection detection algorithm is introduced to detect the collision.

This chapter contains the main solution of the trajectory generation problem for the truck. To use the closed-loop algorithm, we need firstly design controllers for the truck to track the reference paths. In the next chapter, we would like to develop tracking controllers for the truck model.

Chapter 4

Controller Design

The controller takes the motion plan and generates control commands that track the desired motion plan. The motion plan contains the same information as the controller inputs used in the planner prediction, and consists of a set of (X, Y, V_{ref}) points that define the piece-wise affine reference path for the steering controller and the desired speed.

The motion planner contains two controllers, a pure-pursuit steering controller and a PID speed controller. According to the available research, the path-tracking problem of the autonomous vehicle is usually split into the speed control problem and steering control problem [29]. We choose a pure-pursuit steering controller because it shows great tracking performance for robotics, and is widely used for ground and aerial vehicles for many years [30]. The PID controller is used to track the desired speed. These two controllers are not only used in the motion planner, but also used in the execution controller.

4-1 The Control Architecture

The control system of an autonomous vehicle can be represented as in Figure 4-1. It contains a sensing system, a mission planner, a trajectory planner, a vehicle controller, a feedback compensator and the autonomous vehicle. The sensing system consists of a set of sensors. It can capture the environment information and generate a local map. The mission planner tracks the mission state and develops a high-level plan to accomplish the mission. The output of the mission planner is an ordered list of waypoints that are provided to the trajectory planner. The trajectory planner generates a kino-dynamically feasible vehicle trajectory that moves towards the waypoint selected by the mission planner and the information from the local map. The vehicle controller uses the inputs from the feedback compensator and the trajectory planner to execute the low-level control to track the desired paths. The feedback compensator is a component estimating the vehicle states and filtering the noises.

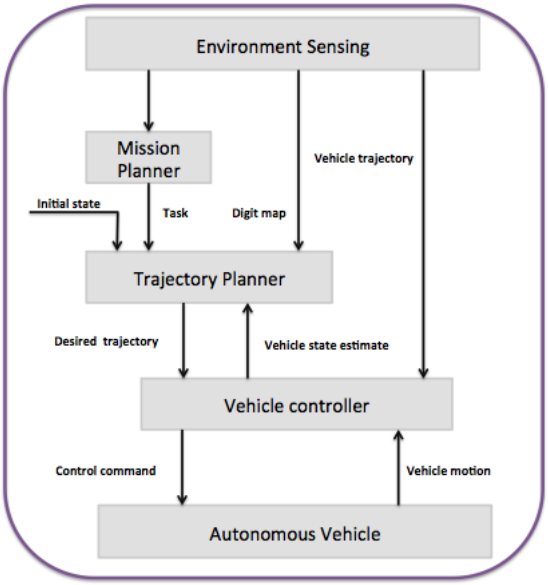


Figure 4-1: The overview of the control system of the autonomous vehicle.

4-2 Speed Controller

The speed controller is designed to adjust the velocity of the vehicle. It works as Figure 4-2 shows. The controller generates the acceleration signal to the actuator when the command velocity is larger than the estimated velocity. It gives braking signal to the executor when the command velocity is lower than the estimated velocity. In this section, the attention will be focused on the speed controller.

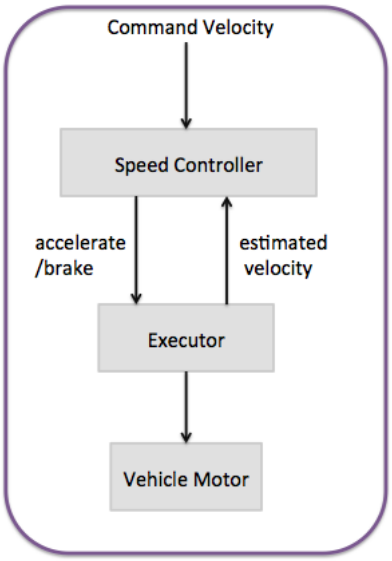


Figure 4-2: The flowchart of a speed controller

4-2-1 PID Controller

A simple proportional-integral-derivative (PID) type controller is the most common form of feedback, and more than 95% of the control loops are of PID type in process control today [31]. The popularity of the PID controller can be attributed to its different characteristic features: it provides feedback; it has the ability to eliminate steady-state offsets through integral action; it can anticipate the future through derivative action; it is easy to use; and it is elegant to understand [29].

The PID controller can be considered as a controller that considers the past, the present, and the expected error. The control signal is thus a sum of three terms: the P-term, the I-term and the D-term. The P-term is proportional to the error and depends on the present error. The I-term is proportional to the integral of the past error. The D-term is proportional to the derivative of the error, and it predicts the future error.

The algorithm of the PID controller is described by Equation 4-1, where u is the control signal, and $e(t)$ is the control error.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{4-1}$$

Transforming Equation 4-1 into the Laplace domain, we can obtain a transfer function of a PID controller. This transfer function of the controller G_c is shown in Equation 4-2.

$$G_c = \frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} + K_d s \tag{4-2}$$

In Equation 4-1 and Equation 4-2, the parameter K_p is the proportional gain, K_i is the integral gain, and K_d is the derivative gain. We can adjust the performance of the PID by tuning these three parameters.

4-2-2 PID Controller Design Method

There are several methods for tuning a PID loop. The most effective methods generally involve the development of some form of process model, then choosing K_p, K_i, and K_d based on the dynamic model parameters. Among these methods, the Ziegler-Nichols design method is the most popular method used in process control to determine the parameters of a PID controller [32]. Besides, this method is a proven method.

The 1st Ziegler-Nichols method is available for the S-shaped reaction curve that characterised by two constants, delay time and time constant, which are determined by drawing a tangent line at the inflection point of the curve and finding the intersections of the tangent line with the time axis and the steady-state level line. However, the open-loop system does not show a S-shaped reaction curve in this project. Hence, we use the 2nd Ziegler-Nichols method to tune the PID loop here.

The 2nd Ziegler-Nichols tuning method is a heuristic method for tuning a PID controller. John G. Ziegler and Nichols proposed it in the 1940's. It is performed by first setting the I-term and D-term gains to zero. The K_p is then increased (from zero) until it reaches the ultimate gain K_u, at which the output of the control loop oscillates with a constant amplitude. K_u and the oscillation period T_u are used to set the K_p, K_i, and K_d gains depending on the type of controller used. The 2nd Ziegler-Nichols tuning principles with regard to different controllers are shown in Table 4-1 [32].

Control Type	K_p	K_i	K_d
P	$0.5K_u$	-	-
PI	$0.45K_u$	$1.2\frac{K_p}{T_u}$	-
PID	$0.6K_u$	$2\frac{K_p}{T_u}$	$\frac{K_pT_u}{8}$

Table 4-1: Ziegler-Nichols Method.

With 2nd Ziegler-Nichols PID tuning formula, the resulting system may exhibit several unacceptable performances, such as a large settling time, slow response speed, and so on. In such a case, we need a series of parameter-selecting until an acceptable result is obtained. The individual effect of K_p, K_i and K_d is listed in Table 4-2 that can be very useful in parameter-selecting of PID controller [7]. Starting with the values of K_i, K_p and K_d obtained from 2nd Ziegler-Nichols method, tuning K_p, K_i and K_d based on Table 4-2 until an acceptable performance is observed.

Parameter	Rise time	Overshooting	Settling time	Steady-state error	Stability
K_i	Decrease	Increase	Increase	Decrease	Degrade
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_d	Small change	Decrease	Decrease	No	Improve with small K_d

Table 4-2: The effects of increasing one parameter K_i, K_p or K_d independently. This table is adopted from [7]

4-2-3 PID Speed Controller Design

The speed controller is designed to make the vehicle tracking the command velocity. A speed control loop is shown in Figure 4-3. The controller sends the accelerating or braking signals to the executor, and then the executor makes corresponding actions to the motor. The motor adjusts its own speed based on the signal from the executor.

According to the previous chapter, the outputs of the executor and the motor can be described as Equation 4-3 and Equation 4-4, respectively.

$$\dot{a}(t) = \frac{1}{T_a}(a_c(t) - a(t)) \tag{4-3}$$

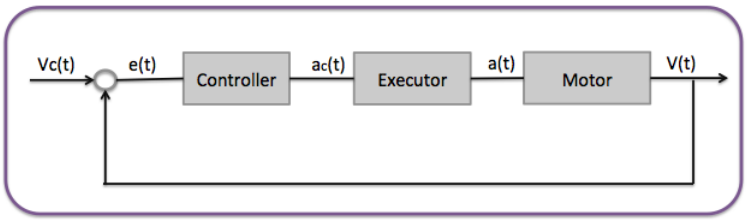


Figure 4-3: The speed control loop

$$\dot{V}(t) = a(t) \tag{4-4}$$

Writing Equation 4-3 and Equation 4-4 in the Laplace domain, and then combining these two Laplace equations, we can obtain the speed transfer function G_v of the vehicle as show in Equation 4-5, where T_a is the delay of the executor and the value of the delay is $T_a = 1.2\ s$.

$$G_v = \frac{V}{a_c} = \frac{1}{(T_a s + 1)s} \tag{4-5}$$

Firstly, we use the Ziegler-Nichols method to tune the closed-loop in Figure 4-3. By setting the I-term and D-term as zeros, we can obtain the closed-loop transfer function G_{cl} of the vehicle in Equation 4-6.

$$G_{cl} = \frac{1}{T_a s^2 + s + K_p} \tag{4-6}$$

According to the Routh-Hurwitz Criterion stability theory, with regard to the second order system, the system is stable if the coefficients of the characteristic polynomial are all positive. Since the K_p is positive, hence the closed-loop system is stable when $K_i = 0$ and $K_d = 0$. That is, the ultimate gain K_u is positive infinite. The step response of the system with $K_p > 0$, $K_i = 0$ and $K_d = 0$ is shown in Figure 4-4.

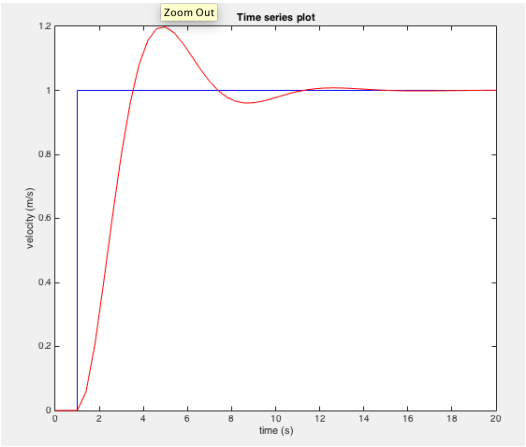


Figure 4-4: The step response of the closed-loop system with parameters $K_p = 1$, $K_i = 0$ and $K_d = 0$.

The overshoot is nearly 20%, the rise time is about 3.5 s, and the settling time is approximately 14 s. In this project, we would like the speed controller could make quick response, and has zero steady-state error. The settling time should be less than 1 s. In order to achieve these goals, a parameter-selecting method is used.

The parameter-selecting method is based on the rules listed in Table 4-2 to tune the PID achieving an acceptable performance. To reduce the rise time, we can increase the value of K_p at first, and then increase the values of K_i and K_d to make the controller perform well. The PID controller offers no significant advantage over the PI controller because the vehicle has some inherent speed damping and the acceleration signal required for PID control is noisy [30]. Figure 4-5 shows that the D-term affects the response of the system slightly. Hence, the PI controller is used here to control the speed.

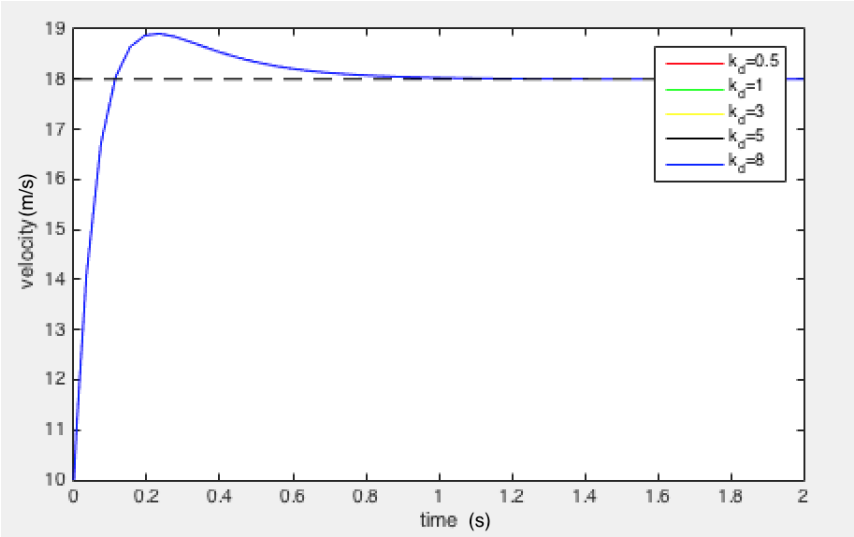


Figure 4-5: The responses comparison of the system with the PI controller and PID controller with different value of K_d. The data is obtained with $K_p = 19.33$ and $K_i = 72.5$, and $K_d = 0.5, 1, 3, 5, 8$.

We have the PI controller with the form of Equation 4-7, where $e(t) = V_{ref} - V$ is the speed difference. The parameters of the PI controller are $K_i = 72.5$ and $K_p = 19.33$. The responses of the PI controller with different values for V_{ref} are illustrated in Figure 4-6.

$$u(t) = K_p(V_{ref} - V) + K_i \int_0^t (V_{ref} - V)d\tau \tag{4-7}$$

In Figure 4-6, the overshoots with regard to speed $V = 12\ m/s$, $V = 15\ m/s$, $V = 18\ m/s$, $V = 20\ m/s$ and $V = 25\ m/s$ are all within 20%, the rise times are all around 0.2 s, and the settling times are all within 0.8 s. The designed PI controller can works well in this situation.

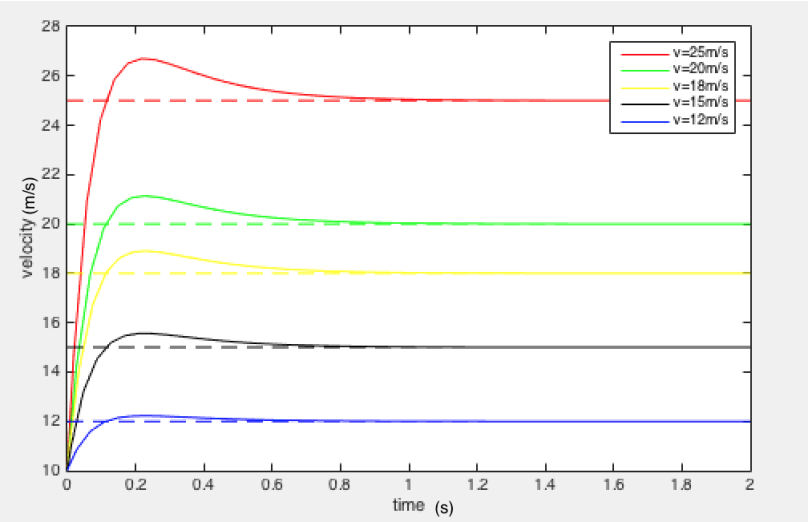


Figure 4-6: The responses of the system with the PI controller. The values of V_{ref} are 12 m/s, 15 m/s,18 m/s, 20 m/s, 25 m/s. $K_i = 72.5$, $K_p = 19.33$.

4-3 Steering Controller

In the previous section, we have designed a PID controller for the velocity control. In this section, we design a steering controller for the truck. A steering control loop is illustrated in Figure 4-7. The steering controller uses the information of the actual path and the reference path to generate the steering command to the steer wheel, then the steer wheel transfers the steering command to the wheels through the executor between them. In this work, we use the pure-pursuit algorithm to design a steering controller for the truck.

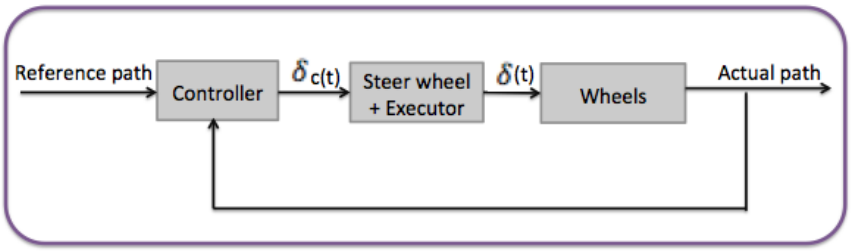


Figure 4-7: The steering control loop.

4-3-1 Pure-Pursuit Control Theory

The pure-pursuit algorithm is a well-known strategy for the mobile robotics community [33]. The steering controller is based on the pure-pursuit control algorithm. The pure pursuit controller is a nonlinear path follower. It is an intuitive control law that is simple and has a clear geometric meaning. The word pure-pursuit implies imagining a vehicle following or chasing a point on given path some distance ahead of it. The principle of the pure pursuit is to calculate the instantaneous curvature of the path that the vehicle intends to generate

with the current speed and heading angle. Figure 4-8 illustrates the detail of pure-pursuit algorithm.

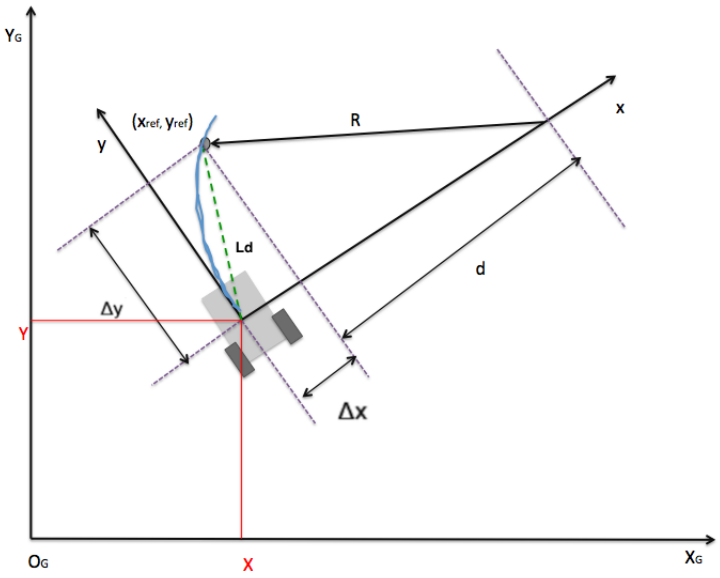


Figure 4-8: The strategy of the pure-pursuit algorithm.

Based on the original description in [34], the implementation of the pure-pursuit algorithm can be summarised as follows:

1. Find the current location of the vehicle in the global frame
2. Find the closest point to the vehicle on the path
3. Find the goal point by choosing a suitable look-ahead distance
4. Transform the goal point in the global frame to the vehicle frame
5. Calculate the curvature and the requested steering angle of the vehicle
6. Update the vehicle's position

Before implementing the algorithm, we present firstly the mathematic description of this algorithm. According to Figure 4-8, we can derive the mathematical equations:

$$\begin{aligned} R &= d + \Delta x \\ R^2 &= d^2 + \Delta y^2 \\ L_d^2 &= \Delta x^2 + \Delta y^2 \end{aligned} \tag{4-8}$$

where R is the turning radius, L_d is the look -ahead distance, Δx and Δy are the coordinates differences between the centre of the vehicle body and the reference points. Eliminating d, and we can obtain the Equation 4-9. Rewriting the Equation 4-9 into the form of Equation 4-10,

and then we can obtain the curvature, as shows in Equation 4-11, which the vehicle should follow.

$$R^2 = (R - \Delta x)^2 + \Delta y^2 \tag{4-9}$$

$$R = \frac{\Delta x^2 + \Delta y^2}{2\Delta x} \tag{4-10}$$

$$\zeta = \frac{1}{R} = \frac{2\Delta x}{\Delta x^2 + \Delta y^2} = \frac{2\Delta x}{L_d^2} \tag{4-11}$$

Using the rotation matrix Equation 2-1, we can transfer the Δx and Δy into the global frame, as Equation 4-12 shows. Now Equation 4-11 can be defined in the global frame as Equation 4-13 illustrates.

$$\begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = R_I^{-1} \begin{bmatrix} \Delta X \\ \Delta Y \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} \Delta X \\ \Delta Y \end{bmatrix} \tag{4-12}$$

$$\zeta_g = \frac{2(cos(\theta)\Delta X + sin(\theta)\Delta Y)}{L_d^2} \tag{4-13}$$

The look-ahead distance L_d plays significant role in the performance of the controller. The look-ahead distance indicates how far along the path the robot should look from the current location to compute the angular velocity commands. It has a significant influence on the performance of the steering control.

Before considering the effects of changing the look-ahead distance, we must distinguish these two conditions: regaining a path (as shows in Figure 4-9) and maintaining the path (as shows in Figure 4-10). When the vehicle is far from the path, it must try to attain on the path. This is the process of the regaining a path. When the vehicle is on the path, it must be try to remain the path. This is the so-called maintaining the path.

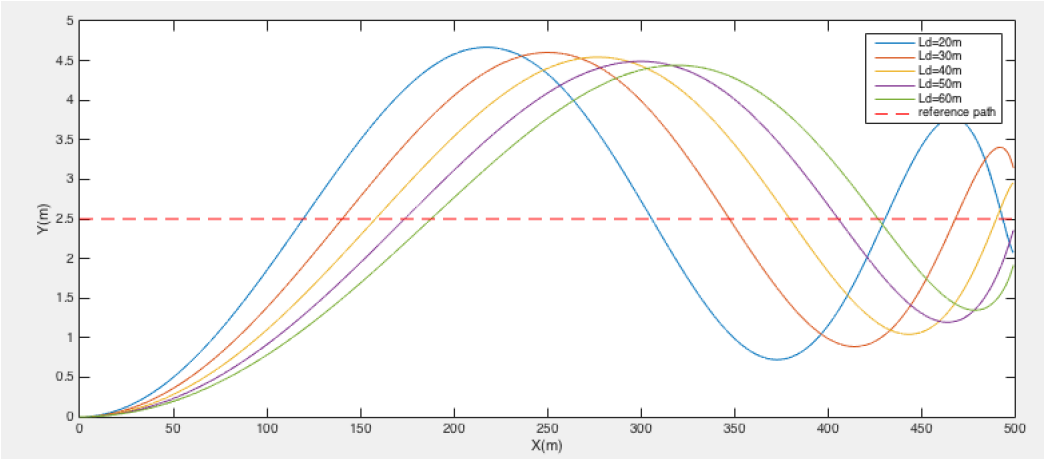


Figure 4-9: The response of the controller with look-ahead distances $L_d = 20\ m$, $L_d = 30\ m$, $L_d = 40\ m$,$L_d = 50\ m$, $L_d = 60\ m$. At first time, the vehicle is not on the path, and then tries to attain the ref path (the red dotted line).

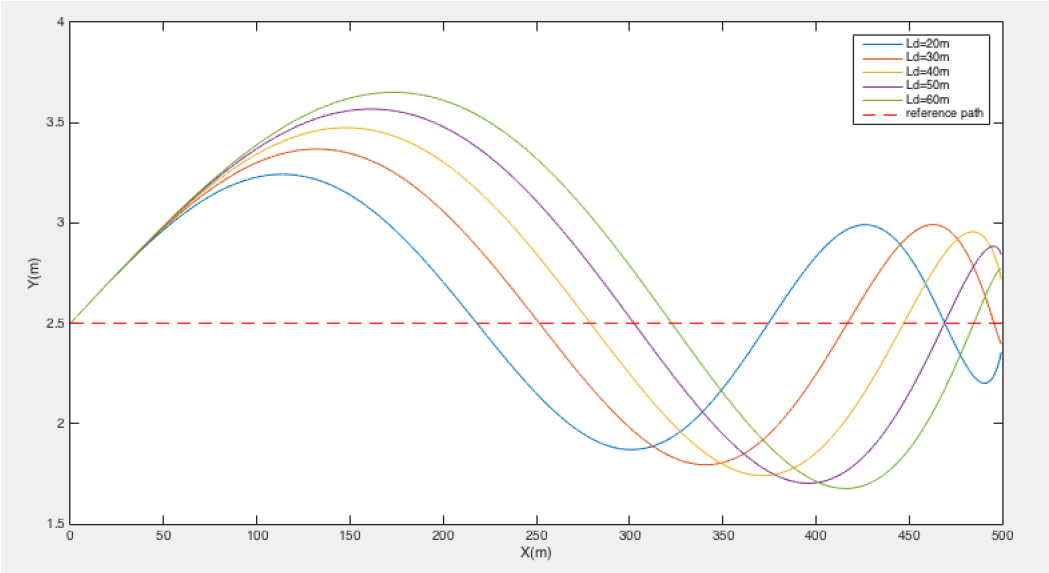


Figure 4-10: The response of the controller with look-ahead distances $L_d = 20\ m$, $L_d = 30\ m$, $L_d = 40\ m$,$L_d = 50\ m$, $L_d = 60\ m$. At first time, the vehicle is on the path, and then tries to maintain the ref path (the red dotted line).

In Figure 4-9 and Figure 4-10, the algorithm tries to fit the angular approaching the reference path. Pure-pursuit is to calculate the instantaneous curvature of the path that the vehicle intends to generate with the current speed and heading angle. The whole point of the algorithm is to choose a goal position that is some distance ahead of the vehicle on the path. We tend to regard the vehicle as following a point on the path some distance ahead of it. That is, we consider the vehicle is pursuing that moving point. It is easy to understand that when the driver can observe long distance ahead of the vehicle at each instantaneous, a small steering angular is needed to drive the vehicle to chase this moving point at this instantaneous. When the driver can observe short ahead of the vehicle at each instantaneous, a large steering angular is needed to drive the vehicle at this instantaneous.

The look-ahead distance L_d is the only parameter of the pure-pursuit algorithm and affects the performance of the steering controller a lot. A long look-ahead distance leads to a smooth converge to the path and less oscillation. A small look-ahead distance tends to quick converge to the path, but with more oscillation. A trade-off between the fast converge and less oscillation must be made when using this algorithm.

4-3-2 Modified Pure-Pursuit Steering Controller

The Massachusetts Institute of Technology (MIT) has developed a modified pure-pursuit control law and analysis the stability of the system with this control law [30]. Instead of using the information of the orientation angle θ, the modified pure-pursuit control law uses the heading angle η and L_r to generate the requested steering angle. Figure 4-11 gives the information about the modified pure-pursuit control law, where L_r is the distance of the forward anchor point from the rear axle, L_d is the forward drive look-ahead distance, L is the length of the wheelbase, η is the the heading of the look-ahead point from the forward anchor point with

respect to the vehicle heading, R is the radius of the curvature, and the reference path is a piecewise affine reference path given by the planner.

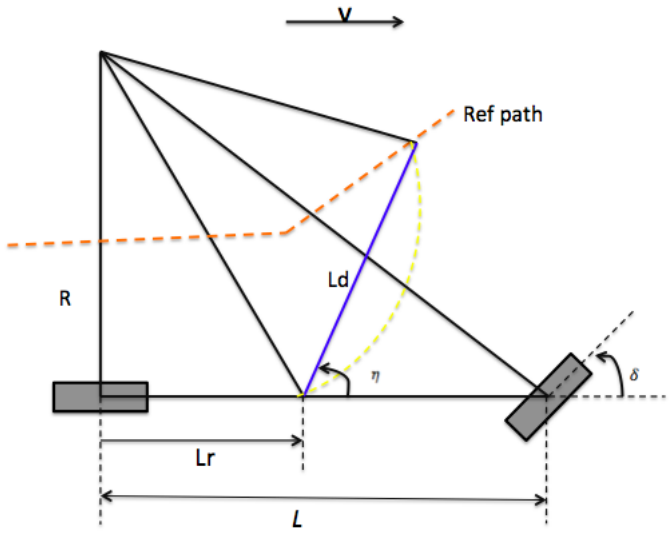


Figure 4-11: The geometric expression of the modified pure-pursuit algorithm. All angles and the lengths in this figure are positive by definition.

The instantaneous steering angle that is requested to put the anchor point on a collision course with the look-ahead points is given by:

$$\delta = -arctan(\frac{Lsin(\eta)}{\frac{L_d}{2} + L_r cos(\eta)}) \tag{4-14}$$

Compared with the conventional pure-pursuit algorithm, the modified one shows a larger relative stability since it introduces the parameter L_r.

Consider the case shows in Figure 4-12. The reference path is a straight line parallel to the $X - axis$ in this case. Based on the geometry, we can get the relation as show in Equation 4-15.

$$\eta = arcsin(\frac{y + L_r sin(\theta)}{L_d}) + \theta \tag{4-15}$$

The vehicle system has a constant maximum slew rate, $-0.1 \leq \dot{\delta} \leq 0.1$. This constraint can be approximated by modelling the actuator dynamics as a first order system (see in Equation 4-16), where δ_c is the steer command, and τ is the actuator time constant [30].

$$\dot{\delta} = \frac{1}{\tau}(-\delta + \delta_c) \tag{4-16}$$

Then the steering control law then can be given by:

$$\delta_c = -arctan(\frac{Lsin(\eta)}{\frac{L_d}{2} + L_r cos(\eta)}) \tag{4-17}$$

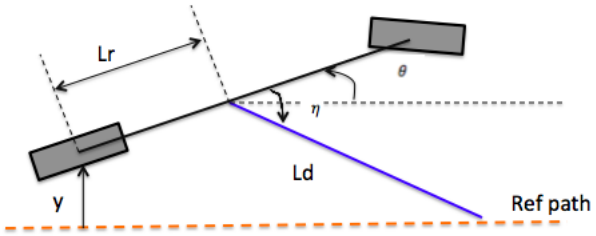


Figure 4-12: The straight reference path along the x-axis.

In order to analyse the stability of the steering closed-loop, we must define a new state $z = [y, \theta, \delta]^T$, and the new state can be expressed in Equation 4-18. By setting $\dot{z} = 0$, we can obtain the equilibrium of the system described in Equation 4-3-2. It is easy to see that the equilibrium is $\hat{z} = [0, 0, 0]^T$.

$$\dot{z} = \begin{bmatrix} \dot{y} \\ \dot{\theta} \\ \dot{\delta} \end{bmatrix} = \begin{bmatrix} Vsin(\theta) \\ \frac{V}{L}tan(\delta) \\ \frac{1}{\tau}(-\delta + \delta_c) \end{bmatrix} \tag{4-18}$$

Linearising the closed-loop system for z, and we can have the linear system as the Equation 4-19 described, where the $\frac{\partial \delta_c}{\partial \eta}$, $\frac{\partial \eta}{\partial y}$ and $\frac{\partial \eta}{\partial \theta}$ can be found in Equation 4-20, Equation 4-21 and Equation 4-22, respectively.

$$\dot{z} = Az = \begin{bmatrix} 0 & Vcos(\theta) & 0 \\ 0 & 0 & \frac{V}{L}sec^2(\delta) \\ \frac{1}{\tau}\frac{\partial \delta_c}{\partial \eta}\frac{\partial \eta}{\partial y} & \frac{1}{\tau}\frac{\partial \delta_c}{\partial \eta}\frac{\partial \eta}{\partial \theta} & -\frac{1}{\tau} \end{bmatrix} z \tag{4-19}$$

$$\frac{\partial \delta_c}{\partial \eta} = -\frac{\frac{LL_dcos(\eta)}{2} + LL_r}{L^2sin^2(\eta) + (\frac{L_d}{2} + L_rcos(\eta))^2} \tag{4-20}$$

$$\frac{\partial \eta}{\partial y} = \frac{1}{\sqrt{L_d^2 - (y + L_rsin(\theta))^2}} \tag{4-21}$$

$$\frac{\partial \eta}{\partial \theta} = \frac{L_rcos(\theta)}{\sqrt{L_d^2 - (y + L_rsin(\theta))^2}} + 1 \tag{4-22}$$

At the equilibrium $\hat{z} = [0, 0, 0]^T$, the linearised system in Equation 4-20 can be rewritten as Equation 4-23. The characteristic equation of the matrix A in the Equation 4-23 can be written in the Equation 4-24.

$$\dot{z} = Az = \begin{bmatrix} 0 & V & 0 \\ 0 & 0 & \frac{V}{L} \\ -\frac{2L}{\tau L_d(L_d+2L_r)} & -\frac{2L(L_d+L_r)}{\tau L_d(L_d+2L_r)} & -\frac{1}{\tau} \end{bmatrix} z \tag{4-23}$$

$$det(sI - A) = s^3 + \frac{1}{\tau}s^2 + \frac{2V(L_d + L_r)}{\tau L_d(L_d + 2L_r)}s + \frac{2V^2}{\tau L_d(L_d + 2L_r)} = 0 \tag{4-24}$$

We use the Routh-Hurwitz Criteria to check the stability. For the third-order polynomial in Equation 4-24, the Routh array is illustrated in Equation 4-25.

$$\begin{array}{c|cc} s^3 & 1 & \frac{1}{\tau} \quad \frac{2V(L_d+L_r)}{\tau L_d(L_d+2L_r)} \\ s^2 & \frac{1}{\tau} & \frac{2V^2}{\tau L_d(L_d+2L_r)} \\ s & \frac{2V(L_d+L_r)}{\tau L_d(L_d+2L_r)} - \frac{2V^2}{L_d(L_d+2L_r)} & \\ s^0 & \frac{2V^2}{\tau L_d(L_d+2L_r)} & \end{array} \tag{4-25}$$

According to the Routh-Hurwitz Criteria [35], the system is stable if and only if:

$$\begin{aligned} \frac{1}{\tau} &> 0 \\ \frac{2V(L_d+L_r)}{\tau L_d(L_d+2L_r)} - \frac{2V^2}{L_d(L_d+2L_r)} &> 0 \\ \frac{2V^2}{\tau L_d(L_d+2L_r)} &> 0 \end{aligned} \tag{4-26}$$

In the merge maneuvers, $V > 0$ and $\tau > 0$. As they are geometric parameters of the truck, $L_d > 0$ and $L_r > 0$. Hence the stability criteria is illustrated in Equation 4-27.

$$L_d > V\tau - L_r \tag{4-27}$$

According to Equation 4-27, the look-ahead distance depends on the speed, actuator time constant τ and the distance from the rear axle to the anchor point. Since both τ and L_r are constant with respect to the same vehicle, the look-ahead distance is mainly determined by the vehicle speed.

4-3-3 Steering Controller Design for the Truck

The modified pure-pursuit steering controller uses the CoG of the vehicle as the anchor point, and we also use the CoG of the truck to define the location of the truck in this work. In this subsection, we design a steering controller for the truck based on the modified pure-pursuit control method.

[30] proposed a scheduled L_d, as Equation 4-28 shown, based on the command velocity. However, the Equation 4-28 is not suitable for the truck since the actuator time constant τ of the truck is larger, and the speed of the truck in the highway is higher than the speed of the vehicle in the urban situation.

$$L_d(V_{ref}) = \begin{cases} 3 & V_{ref} < 1.34m/s \\ 2.24V_{ref} & 1.43m/s \leq V_{ref} < 5.36m/s \\ 12 & otherwise \end{cases} \tag{4-28}$$

Although the Equation 4-28 does not work for the truck in highway, it gives us good guidance. Since this work is focused on merging manoeuvres of the truck on the highways, we only consider the velocity in these manoeuvres. The [36] has pointed out that the average speed of the truck in a four-lane divided highway is around 18.3 m/s (ca. 66 km/h). Hence, we can

assume that the velocity of the truck does not exceed 18.3 m/s during merging manoeuvres. According to Equation 4-28, L_d could be chosen as 12 m. To ensure the stability, Equation 4-27 must be satisfied. The actuator time constant of truck is $\tau = 1.5$ s, the minimum value of L_d with respect to different speeds are listed in Table 4-3

V_{ref}	11.1 m/s (ca. 40 km/h)	18.3 m/s (ca.66 km/h)	27.8 m/s (ca.100 km/h)
$L_{d,min}$	14.2 m	25 m	39.2 m

Table 4-3: The minimum value of the look-ahead distance L_d with different velocities.

In order to keep the stability of the controller during merging manoeuvres, the minimum value of the look-ahead distance L_d cannot smaller than 25 m. We have already known that a larger L_d means smoother converge to the reference path. Figure 4-13 shows the tracking performance of the steering controller with different values of the look-ahead distance L_d. In Figure 4-13, the controller with $L_d = 30$ m tracks the reference paths best and shows least tracking error. Hence, we can choose the $L_d = 30$ m in this work.

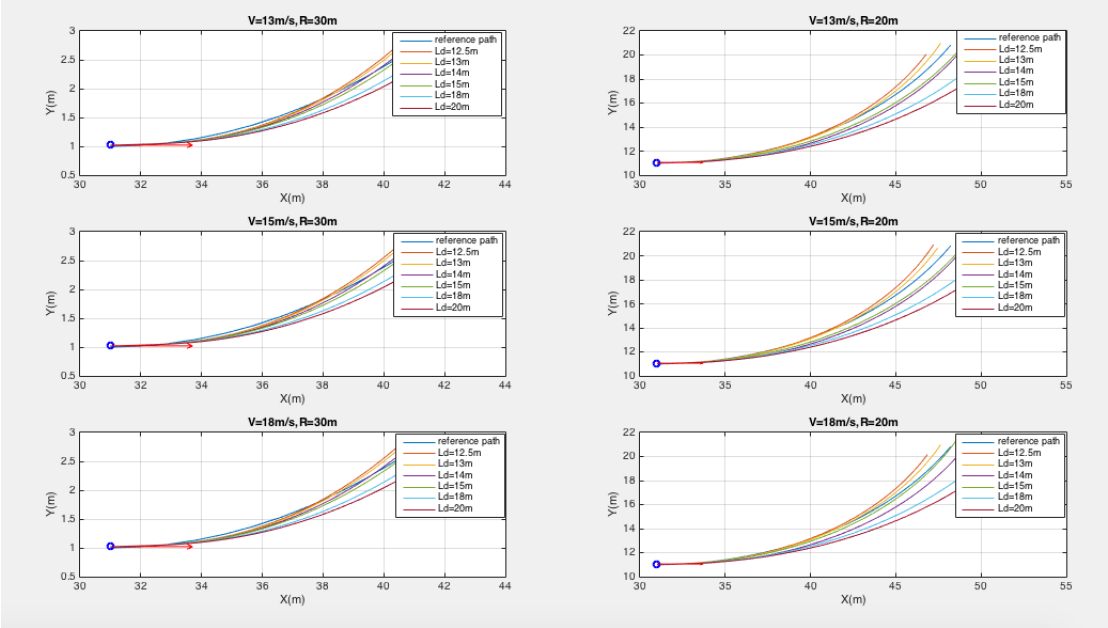


Figure 4-13: The performance of the steering controller with different values of L_d.

To test the designed steering controller, we regard the velocity as a constant. Figure 4-14 shows the tracking performance of the steering controller with $V = 18$ m/s. Figure 4-15 shows the tracking performance of the steering controller with $V = 13$ m/s From Figure 4-14 and Figure 4-15, it can be seen that the steering controller can track the reference curve with different curvatures. There exist tracking errors, but the tracking errors are small.

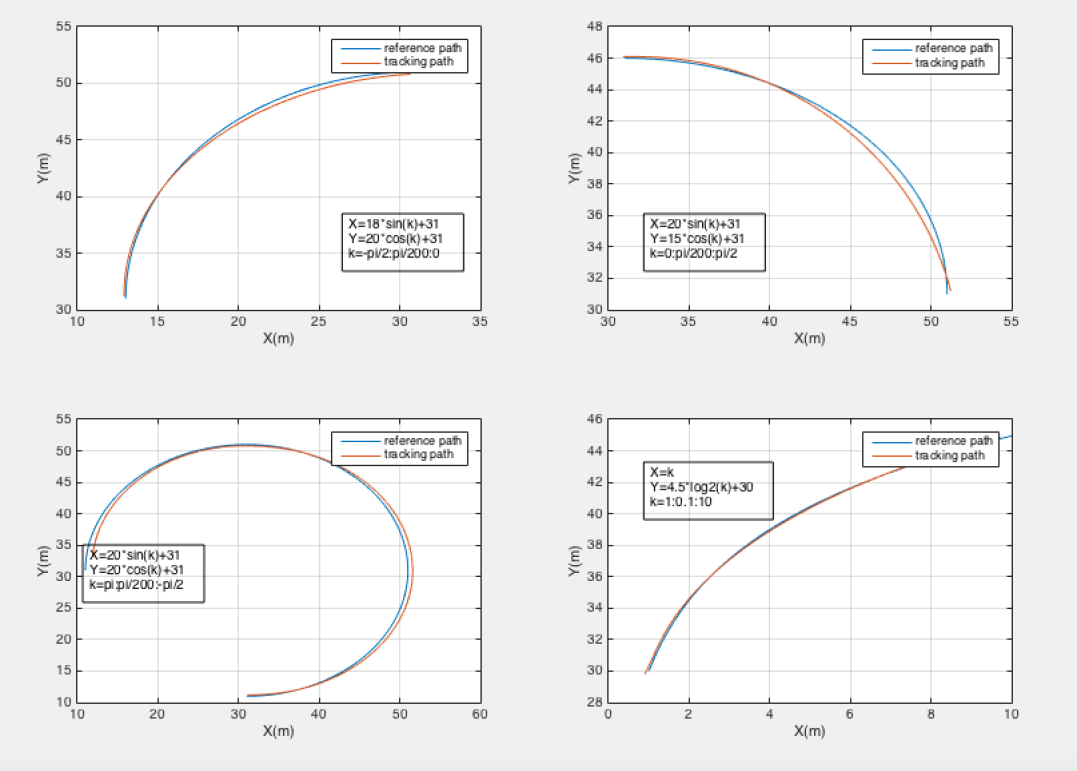


Figure 4-14: The performance of the controller with the modified pure-pursuit algorithm. The look-ahead distances L_d is chosen as $L_d = 30\ m$. The velocity $V = 18\ m/s$. The blue line is the reference curve, the red line is the tracking curve.

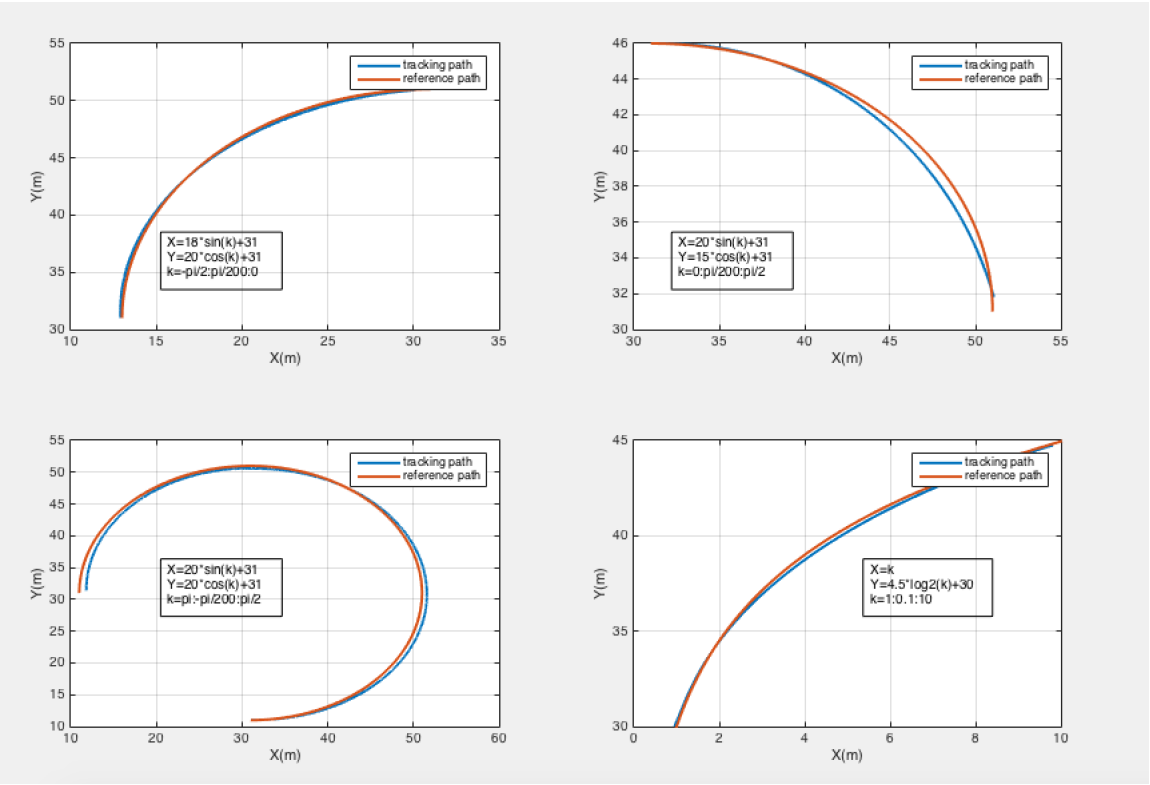


Figure 4-15: The performance of the controller with the modified pure-pursuit algorithm. The look-ahead distances L_d is chosen as $L_d = 30\ m$. The velocity $V = 13\ m/s$. The blue line is the reference curve, the red line is the tracking curve.

4-4 Summary

The controller is designed to make the vehicle following the reference path. In this chapter, we focus on the controller designed for controlling the speed and the steer angle.

A PID controller is designed here to control the speed of the truck. When the truck merges from the ramp to the highway, it should have a similar velocity to the vehicles in the target lane. The designed PID controller can make quick response and zero steady-state errors. Moreover, it can keep the speed of the truck in a range from 12m/s to 25m/s, which satisfies the requirements of merging manoeuvres.

A pure-pursuit controller is designed to control the steering of the truck. A pure-pursuit controller is a simple control law with a clear geometric meaning. It works by calculating the instantaneous curvature of the path that the vehicle intends to generate with the current state. The look-ahead distance imposes a significant effect on the performance of this kind of algorithm. A large look-ahead distance brings fewer disturbances to the system, but it also needs more time to converge to the reference path. The MIT proposed a modified pure-pursuit control law. This control law uses the anchor point instead of the rear point to calculate the instantaneous steering angle, which provides a large stable range for the controller. The minimum value of the look-ahead distance are pointed out in Table 4-3. In this case, we choose $L_d = 30\ m$ to design the steering controller. The result shows that the designed modified pure-pursuit controller can track the reference curve with a small tracking error.

These two controllers are designed for the truck model in the trajectory generator. Usually, the truck uses the same controllers to track the outputs of the generator. There exists a mismatch between the truck model and the actual truck, therefore, the parameters of the designed controllers may need to be adjusted when the controllers are used in the actual truck.

Chapter 5

Simulations and Results

The previous chapter has introduced the solutions of the trajectory generation problem. In this chapter, we focus on the implementations of the algorithms described in previous chapter.

The RRT algorithm works in the configuration space. Before implementing the algorithm, we should model the road and the traffics information into the configuration space. In this work, the road is represented in the configuration space by using the coordinate frame like Figure 5-1. The locations of vehicles are described in the configuration space by using the coordinates of their CoGs.

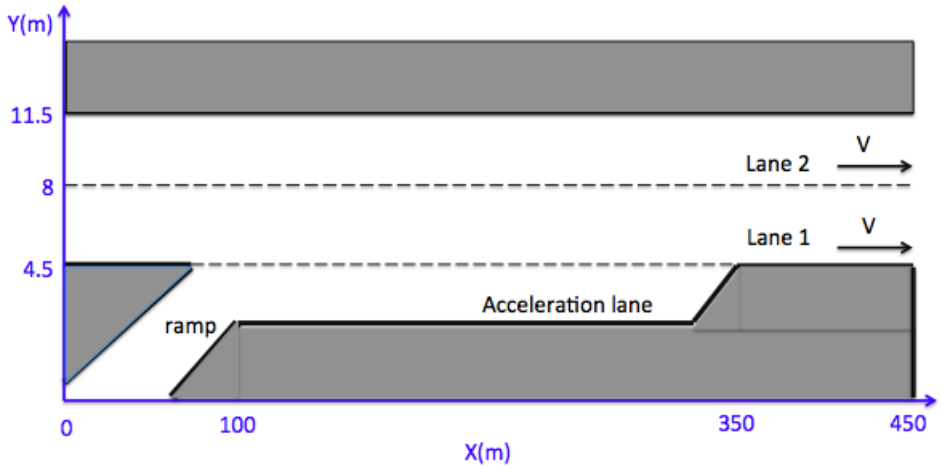


Figure 5-1: The road in the configuration space. The width of each lane is 3.5 m and the total length of the road is 450 m. The grey area is the infeasible area that the truck cannot enter into, and the white area is the drivable area.

The main ingredients of the planner are the start state, goal state and the deviations for the goal state. The start state is the current state of the vehicle when the merging command is

given. The goal state is the ideal state we want to achieve. Usually, it is time-consuming to achieve an ideal state. To speed up the algorithm, we define an acceptable deviation for the goal state. This is the so-called goal region that is an area centred by the goal point. It is determined when goal point and the deviations are given. The algorithm begins to work after the merging command is given, and stops when the goal region is reached. As the previous chapter discussed, with regard to the merging manoeuvres, the centre deviation of the truck and the lane cannot exceed 0.5 m in Y-axis for the purpose of safety. Compared with the deviation of the goal point in the Y-axis, the deviation in X-axis could be larger since we do not require the truck arriving at a specific point in the X-axis. Before we implement the algorithm in different scenarios, these ingredients must be determined.

The algorithm is implemented in MATLAB/SIMULINK environment in this work. There are two key time steps: the path update time-step Δt and the integration time-step ΔT. For the offline implementation over open-loop system, the truck model is solved by the Runga-Kutta method with time step size $\Delta T = 0.03$ s and $\Delta T = 0.05$ s, respectively. For the implementations over closed-loop system, the truck motion is analysed in SIMULINK. There are two types of solvers in SIMULINK: the variable-step solvers and the fixed-step solvers. Fixed-step solvers solve the model at step sizes from the beginning to the end of the simulation, whereas variable-step solvers vary the step size during the simulation. The choice between these types depends on how to deploy the model and the model dynamics. If planning to generate code from your model and run the code on a real-time computer system, a fixed-step solver is chosen to simulate the model, since it is impossible to map the variable-step size to the real-time clock. If we want to solve the truck motion in short time, a variable-step solve is recommended.

In order to implement the algorithm in real-time condition, we use a fix-step solver to solve the model in real-time implementation. The step size of the solve affects the efficiency a lot. The integration time-step is set as $\Delta T = 0.001$ s in this work, since we want a good accuracy of the results. In the highway scenarios, the velocities of the surrounding traffics are high. To guarantee the safety, the update time-step cannot be large. In this case, we set it as $\Delta t = 0.05$ s. The algorithm extends until the environment information is updated, then the tree is cleared and a new tree is built based on the new information of the environment.

The algorithm is mainly implemented in the merging scenarios with different parameters. Considering a basic merging scenario, as Figure 5-2 shows, the truck moves from the acceleration lane to the target lane, and plans to merge into the putative leader (PL) vehicle and the putative follower (PF) vehicle on the highway. The task of the generator is to plan a trajectory for the truck merging into the gap between putative leader (PL) vehicle and the putative follower (PF) vehicle. The red dotted block in Figure 5-2 is the goal region. Its size is determined by the tolerance of the final state, and its location is determined by the goal point. The scenarios are based on this simple scenario in this chapter.

This chapter contains four sections. The first section shows the results of the bias for the sampling strategy. The second section shows the results of the offline implementation over open-loop system and closed-loop system, and the discussion is made under the results. The third section illustrates the results of the on-line implementation with different gaps and velocities in the merging scenario, and the discussion is also made under these results. The last section summaries the main work of this chapter.

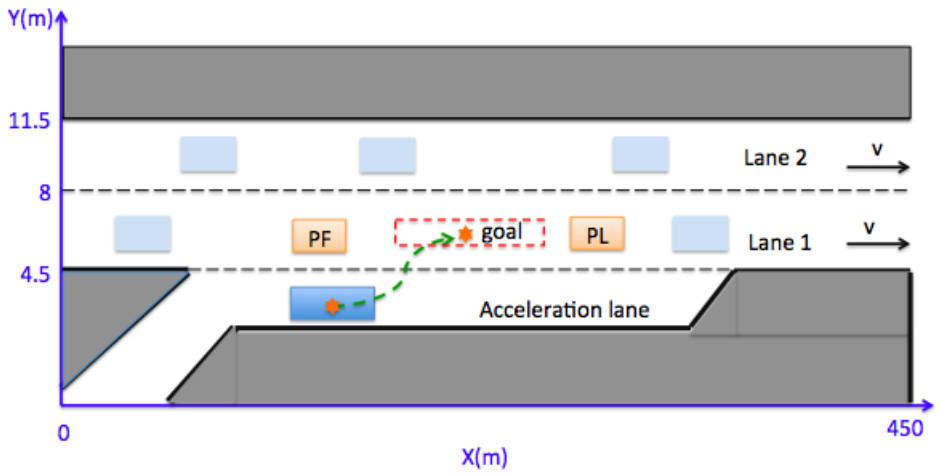


Figure 5-2: The merging scenario of a truck. The orange blocks represent the putative leader vehicle and the putative follower vehicle. The dark blue block represents the truck. The red stars represent the start point and the goal point. The red dotted block is the goal region.

5-1 Parameterisation of Bias

The bias that is used to change the sampling probabilities of the points in the configuration space is the key of the sampling strategy. In this section, the bias is determined by testing the algorithm with different biases in the merging scenario. As the previous chapter described, the standard Gaussian distributions is used in this work. The parameters of the bias that needs to be defined in Equation 3-2 are the offset of the goal region r_0, the standard deviation σ_r in the radial direction and σ_θ in the circumferential direction. The Bias is determined over the closed-loop system offline. The scenarios used in this section are described as follows.

Considering the configuration space as Figure 5-3 shows, the gaps between two vehicles are 100 m in the Lane 1, and are 150 m in the lane 2 in this configuration space. The state of the truck can be described by (X, Y, θ, V) at time t, where (X, Y) is the coordinates of the CoG of the truck, θ is the orientation angle and the V is the speed of the truck. To simplify the problem, we first assume that the truck moves with constant velocity. Varying speeds will be addressed in next paragraph. Hence, the state of the truck is (X, Y, θ) in this section. Since we always expect the truck can arrive at the centre of the lanes in the Y-axis, we use 2.75 m, 6.25 m, and 9.75 m as the coordinate of Y-axis for the vehicles located in the acceleration lane, Lane 1 and Lane 2 in Figure 5-3, respectively. As the aim is to determine the parameters of the bias and to prove the advantage of the sampling strategy, we set the tolerance of the goal $[\Delta X, \Delta Y, \Delta\theta]$ as $[30\ m, 0.25\ m, 0.1\ rad]$ in the configuration space.

To prove the advantage of the sampling strategy, a comparison has been made between the algorithm with the sampling strategy and without this strategy in five different scenarios. In the first two scenarios, the truck merges from the acceleration lane to Lane 1. In the scenario 3 and scenario 4, the truck changes its lane from the Lane 1 to the Lane 2. In the last scenario, the truck follows the leader vehicle in the Lane 1. In order to make comparison more easily, we assume the truck moves with constant speed 16.7 m/s (ca.60 km/h) in these five

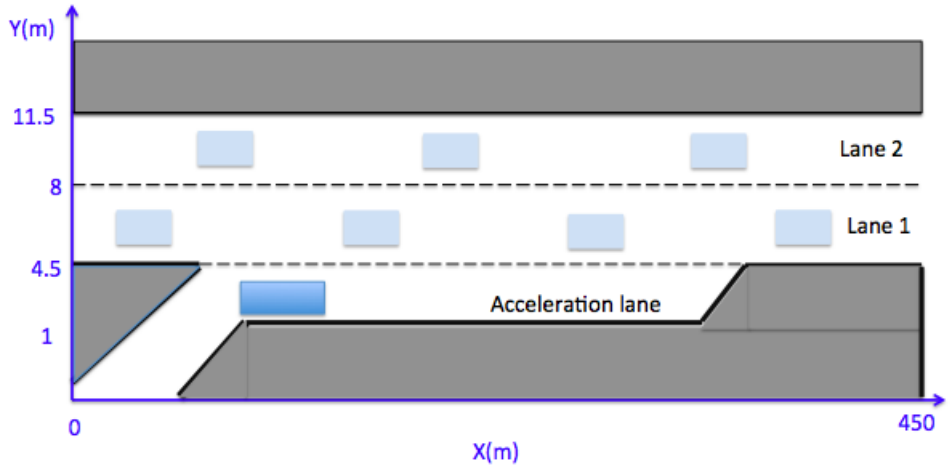


Figure 5-3: The configuration space used in this sections. The gaps between two vehicles are 100 m in the Lane 1, and are 150 m in the lane 2 in this surface. The dark blue block represents the truck, the light blue blocks present the surrounding traffics.

scenarios. The time duration are listed in Table 5-1. The parameters used in this comparison are listed in Table 5-2.

Scenario	1	2	3	4	5
Start state (X, Y, θ)	$(100, 2.75, 0)$	$(150, 2.75, 0)$	$(160, 6.25, 0)$	$(170, 6.25, 0)$	$(350, 6.25, 0)$
Goal state (X, Y, θ)	$(160, 6.25, 0)$	$(210, 6.25, 0)$	$(220, 9.75, 0)$	$(230, 9.75, 0)$	$(410, 6.25, 0)$
Time with bias	13.617 s	14.749 s	14.760 s	15.796 s	16.076 s
Time without bias	14.513 s	20.093 s	19.486 s	19.599 s	55.767 s
Generated nodes with bias	80	83	108	114	115
Generated nodes without bias	100	132	147	148	442

Table 5-1: Comparison the costs of the algorithm with bias and without bias. The costs are described by the needed time to achieve the goal, and the nodes in the tree. The tests are implemented with the same conditions. The planned paths for these five scenarios consist of nearly the same number of nodes.

σ_θ (rad)	σ_r (m)	$r_0(m)$	$\theta_0(rad)$	μ_θ (rad)	μ_r (m)
$\frac{\pi}{12}$	3	1	0	0	0

Table 5-2: Parameters for the bias to the distributions used in comparison between the algorithm with bias and without bias.

From Table 5-1, we can see that the RRT algorithm with bias spends less time to achieve the goal than the algorithm without bias. The nodes needed to plan the path are less for the algorithm with bias than for the algorithm without bias. More useless nodes are generated by the algorithm without bias. The planned paths for these five scenarios consist of almost the same number of nodes.

Adding a bias to the algorithm can improve the efficiency of the algorithm, but different biases impose different effects on the result. The parameters of Equation 3-2 are determined by the vehicle location, road condition and the traffic rules. For the merging manoeuvres on the highway, the vehicle can only move forward in a lane. So the nodes located in the moving forward direction need to be sampled with high probability. In order to make the distribution easy to understand, we set $\mu_r = 0$ and $\mu_\theta = 0$. Since the bias is determined for merging manoeuvres, we can define $\theta_0 = 0$ at initial state.

Firstly, we determine the value of the offset of the goal point r_0. We keep the values of σ_r and σ_θ as constant and change the value of r_0, then compare the performance of the algorithm. Table 5-3 depicts the results of the time, the planned path and the number of generated nodes with different values of r_0. From Table 5-3, we can see the value of r_0 imposes small influences on the results when the value of r_0 is not large. In this project, we determine $r_0 = 1\ m$.

σ_r (m)	σ_θ (rad)	r_0 (m)	Time (s)	Nodes of path	Generated nodes
$\frac{\pi}{12}$	3	0.01	19.98	55	56
$\frac{\pi}{12}$	3	0.1	19.96	55	56
$\frac{\pi}{12}$	3	0.5	18.95	55	56
$\frac{\pi}{12}$	3	1	19.01	55	56
$\frac{\pi}{12}$	3	3	19.13	55	57
$\frac{\pi}{12}$	3	5	20.12	55	57
$\frac{\pi}{12}$	3	10	60.65	60	132
$\frac{\pi}{8}$	3	0.01	21.24	60	61
$\frac{\pi}{8}$	3	0.1	21.27	60	62
$\frac{\pi}{8}$	3	0.5	22.74	63	65
$\frac{\pi}{8}$	3	1	22.23	62	63
$\frac{\pi}{8}$	3	3	22.91	63	65
$\frac{\pi}{8}$	3	5	24.58	65	70
$\frac{\pi}{8}$	3	10	181.75	60	214

Table 5-3: The comparison results of the costs with different value of r_0. $\sigma_\theta = \frac{\pi}{12}$, $\sigma_r = 3$, $\mu_r = 0$ and $\mu_\theta = 0$.

Figure 5-4 illustrates the distribution of the points with different parameters. From Figure 5-4, it can be concluded that a longer distribution can be obtained by increasing the σ_r, and a wider distribution can be obtained by increasing the σ_θ.

The values of σ_r and σ_θ affect the efficiency of the sampling. With respect to merging manoeuvres on the highway, the speed varies from 11.1 m/s (ca.40 km/h) to 27.8 m/s (ca.100 km/h) for the truck, and more than 70% drivers begin to merge into the main lane at the first two third part of the acceleration lane [37]. Hence, we choose the points $(150, 2.75)$ and $(210, 6.25)$ as the start point and the goal point, respectively, to test the costs of the algorithm with respect to different values of σ_θ and σ_r. The initial and final speed of the truck can set as $V_{init} = 16.7\ m/s$ (ca.60 km/h) and $V_{final} = 18.3\ m/s$ (ca.66 km/h), respectively. Table 5-4 lists the results of these tests.

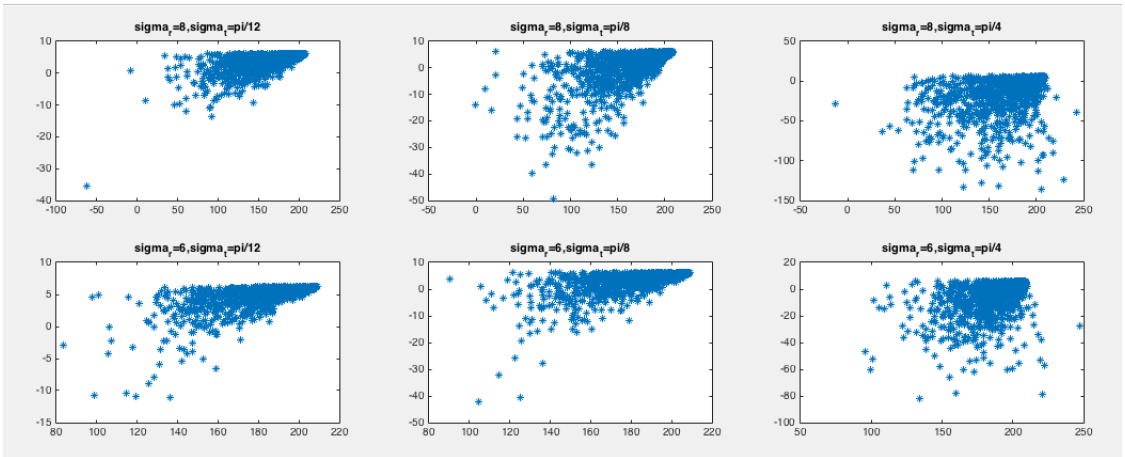


Figure 5-4: The distribution of 500 points with different standard deviation σ_r and σ_θ, the means μ of the Gaussian distributions are $\mu = 0$.

According to Table 5-4, we can see that the algorithm with the bias $\sigma_r = 3\ m$ and $\sigma_\theta = \frac{\pi}{12}\ rad$, $\sigma_r = 3\ m$ and $\sigma_\theta = \frac{\pi}{8}\ rad$, $\sigma_r = 1\ m$ and $\sigma_\theta = \frac{\pi}{8}\ rad$, $\sigma_r = 0.1\ m$ and $\sigma_\theta = \frac{\pi}{4}\ rad$ and $\sigma_r = 0.01\ m$ and $\sigma_\theta = \frac{\pi}{4}\ rad$ use similar time to achieve the goal. The time to reach the goal is around 20 s. Among them, the algorithm with $\sigma_r = 1\ m$ and $\sigma_\theta = \frac{\pi}{8}\ rad$ use least time, but the algorithm with $\sigma_r = 3$ and $\sigma_\theta = \frac{\pi}{8}$ generates longest path. A longer path means the algorithm can look ahead farther. Hence, we choose $\sigma_r = 3$ and $\sigma_\theta = \frac{\pi}{8}\ rad$ as the parameters of the bias for merging manoeuvres in this work. Table 5-5 gives all the information of the bias used in this work. Figure 5-5 shows the distribution with the parameters in Table 5-5.

In addition, from Table 5-4 we can find that the algorithm can achieve the goal with $\sigma_r = 11$

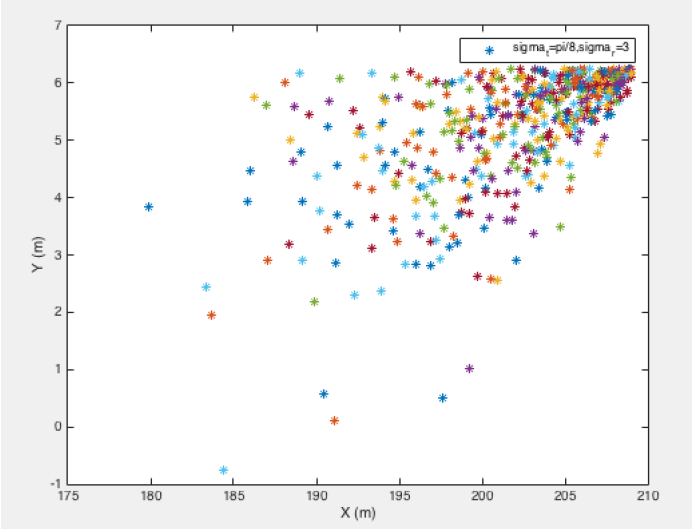


Figure 5-5: The distribution with the parameters listed in Table 5-2. 500 samples have been plotted in this figure. The goal point is $(210, 6.25)$

m when $\sigma_\theta = \frac{\pi}{12}\ rad$; the algorithm can achieve the goal with $\sigma_r = 1\ m$ when $\sigma_\theta = \frac{\pi}{8}\ rad$;

σ_r (m)	σ_θ (rad)	Time (s)	Nodes of path	Generated nodes
11	$\frac{\pi}{12}$	49.41	45	116
11	$\frac{\pi}{8}$	727.27	70	1896
11	$\frac{\pi}{4}$	760.84	70	2034
9	$\frac{\pi}{12}$	33.66	47	95
9	$\frac{\pi}{8}$	69.47	61	173
9	$\frac{\pi}{4}$	616.68	70	1569
7	$\frac{\pi}{12}$	38.63	63	110
7	$\frac{\pi}{8}$	59.10	63	140
7	$\frac{\pi}{4}$	454.34	70	1087
5	$\frac{\pi}{12}$	22.21	55	68
5	$\frac{\pi}{8}$	30.37	62	91
5	$\frac{\pi}{4}$	464.33	67	1159
3	$\frac{\pi}{12}$	20.13	55	56
3	$\frac{\pi}{8}$	20.92	63	64
3	$\frac{\pi}{4}$	59.24	70	71
1	$\frac{\pi}{12}$	110.71	55	56
1	$\frac{\pi}{8}$	19.24	57	58
1	$\frac{\pi}{4}$	25.38	64	66
0.5	$\frac{\pi}{12}$	>2000	–	–
0.5	$\frac{\pi}{8}$	49.32	57	57
0.5	$\frac{\pi}{4}$	22.09	63	64
0.1	$\frac{\pi}{12}$	>2000	–	–
0.1	$\frac{\pi}{8}$	>2000	–	–
0.1	$\frac{\pi}{4}$	20.38	60	62
0.01	$\frac{\pi}{12}$	>2000	–	–
0.01	$\frac{\pi}{8}$	>2000	–	–
0.01	$\frac{\pi}{4}$	19.41	60	65

Table 5-4: The comparison results of the costs with different value of σ_θ and σ_r. The $\mu_r = 0$ and $\mu_\theta = 0$. The symbol '–' means no nodes is generated.

σ_θ (rad)	σ_r (m)	$r_0(m)$	$\theta_0(rad)$	μ_θ (rad)	μ_r (m)
$\frac{\pi}{8}$	3	1	0	0	0

Table 5-5: Parameters for the bias to the distributions of merging maneuvers on the highway.

and the algorithm can achieve the goal with $\sigma_r = 0.5\ m$ when $\sigma_\theta = \frac{\pi}{4}\ rad$. Although the listed data are limited, they can still show a tendency that the algorithm works efficiently with a large σ_r and a small σ_θ.

5-2 Offline Implementation

The off-line RRT algorithm is implemented when the information of the environment is not updated. We apply this algorithm on both the open-loop system and the closed-loop system in this section.

In this scenario, the vehicle try to merge into two vehicles in the lane, as Figure 5-2 shows, and the gap between these two vehicles is 100 m. The tolerance of the goal $[\Delta X, \Delta Y, \Delta\theta]$ is set as $[30\ m, 0.2\ m, 0.15\ rad]$ for the implementation over the open-loop system. With regard to the closed-loop system, both the tolerances of $[40\ m, 0.2\ m, 0.15\ rad]$ and $[20\ m, 0.1\ m, 0.05\ rad]$ are used.

The first part of this section discusses the algorithm implementation in the open-loop system. The middle of this section discusses the algorithm implementation in the closed-loop system. At the end of this section, the results are discussed.

5-2-1 Offline Implementation over the Open-Loop System

The open-loop vehicle system is described in Equation 2-8. The configuration space C and the inputs space U to the open-loop system are:

$$[X, Y, \theta, V, \delta, \dot{V}, \dot{\delta}]^T \in C$$
$$[a, r]^T \in U$$

Here, we only use the sampling strategy for the sampling of configuration space in order to make clear comparison with the implementation over closed-loop system. As Equation 3-4 illustrates, the step-size ΔT affects the results of the Runge-Kutta method. A suitable integration step-size ΔT affects the smoothness of the trajectory and the computation time. If the time step-size is too large, each step length will be too long. A long trajectory means high collision probability and low smoothness. However, if the time step size is too small, the computation time increases. Table 5-6 has listed the computation time of the algorithm to find a feasible path with different time step sizes, the generated nodes and the nodes of the planned path.

In Table 5-6, the computation time of the algorithm with step-size $\Delta T = 0.03\ s$ to find the path is as 8 times as the time of the algorithm with step-size $\Delta T = 0.05\ s$. However, they generate paths with same length. The total number of generated nodes for the algorithm with step-size $\Delta T = 0.03\ s$ is as about 2.3 times as the total number of generated nodes for the algorithm with step-size $\Delta T = 0.05\ s$. The planned the paths and their corresponding

ΔT	Time(s)	Generated Nodes	Nodes of path
0.05	10.9	3896	31
0.03	89.7	8976	31

Table 5-6: The computation time of the algorithm to find a feasible path with different time step sizes. The generated nodes means the nodes in the tree, and the nodes of path means the number of the nodes constructing the planned path.

parameters are presented in Figure 5-7 and Figure 5-6.

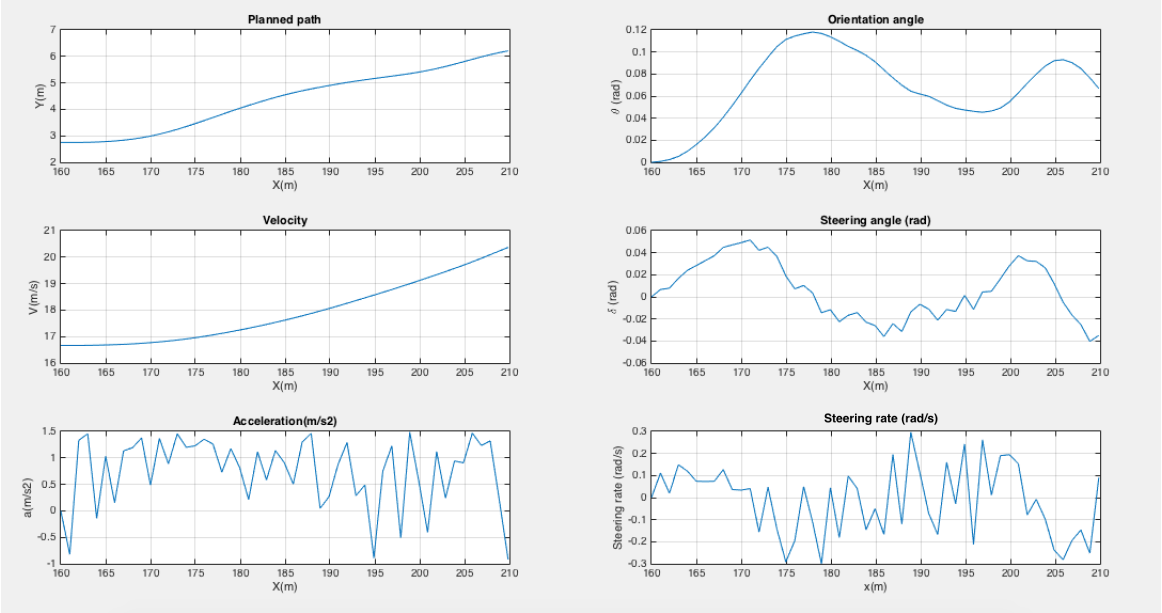


Figure 5-6: The generated path and its corresponding velocity, steering angle, orientation angle, steering rate and acceleration. The step-size $\Delta T = 0.03\ s$.

In Figure 5-6 and Figure 5-7, we can see that the accelerations and steering rates have obvious disturbances. The main reason for the disturbances is that the acceleration and the steering rate are randomly sampled from the inputs space U. The disturbances may be reduced by sampling the derivatives of the acceleration and steering rate, instead of sampling the acceleration an steering rate directly. The two paths also show some disturbances. The path planned with step size $\Delta T = 0.03\ s$ is much smoother than the path planned with step size $\Delta T = 0.05\ s$.

5-2-2 Offline Implementation over the Closed-Loop System

The vehicle model of the closed-loop system is described in Equation 2-14. The configuration space C and the inputs space U to the closed-loop system are:

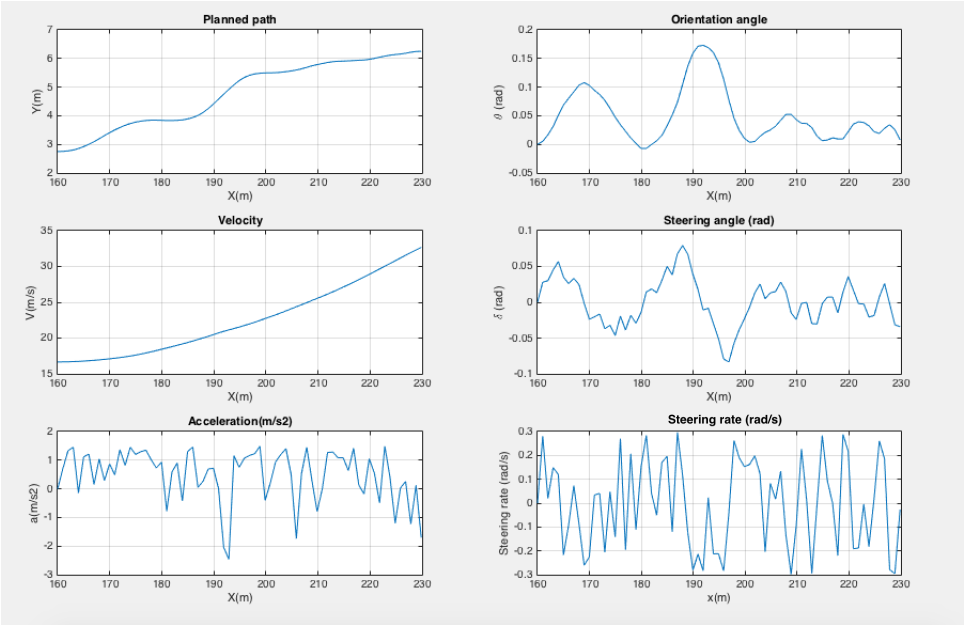


Figure 5-7: The generated path and its corresponding velocity, steering angle, orientation angle, steering rate and acceleration. The step-size $\Delta T = 0.05\ s$.

$$[X, Y, \theta, V, \delta, \dot{V}, \dot{\delta}]^T \in C$$
$$[X_{ref}, Y_{ref}]^T \in U$$

Using the closed-loop RRT algorithm illustrated in Algorithm 3, we can plan a trajectory for the truck in the scenario described in Figure 5-3. The path generated by the closed-loop planner is shown in Figure 5-8. The lateral deviation is 0.15 m and the final orientation angle is 0.09 rad. The reference paths and the generated path are illustrated in Figure 5-9.

The closed-loop RRT algorithm samples the input to the controller, and we use the same controllers to control the lower level system. Hence, it can validate the planned trajectory by reconstructing a trajectory with the reference path in a closed-loop system. Figure 5-10 compares the reconstructed trajectory, the planned trajectory and the reference paths used to reconstructed trajectory. From Figure 5-10, we can see that there exists a small error between the reconstructed trajectory and the planned trajectory. This is because the RRT algorithm is a discrete method to plan the trajectory. The planned path consists of a series of short line segments, whereas the reconstructed path is a continuous path. An optimisation algorithm may reduce the error, but it cannot eliminate the error.

The length of the step size Δr in the function $STEER()$ in Algorithm 2 imposes an effect on the closed-loop RRT algorithm. Increasing the Δr means increasing the length of the reference path. The longer reference path means a larger step size which will lead to a large quantity of waste samples. When we set $\Delta r = 1\ m$, the trajectories in Figure 5-8 can be obtained, however, when we enlarge Δr to $\Delta r = 3\ m$, no trajectory can be found. Nevertheless, it does not mean that a smaller Δr results in higher efficiency. When we set $\Delta r = 0.5\ m$, a trajectory is planned with thousands seconds computation time.

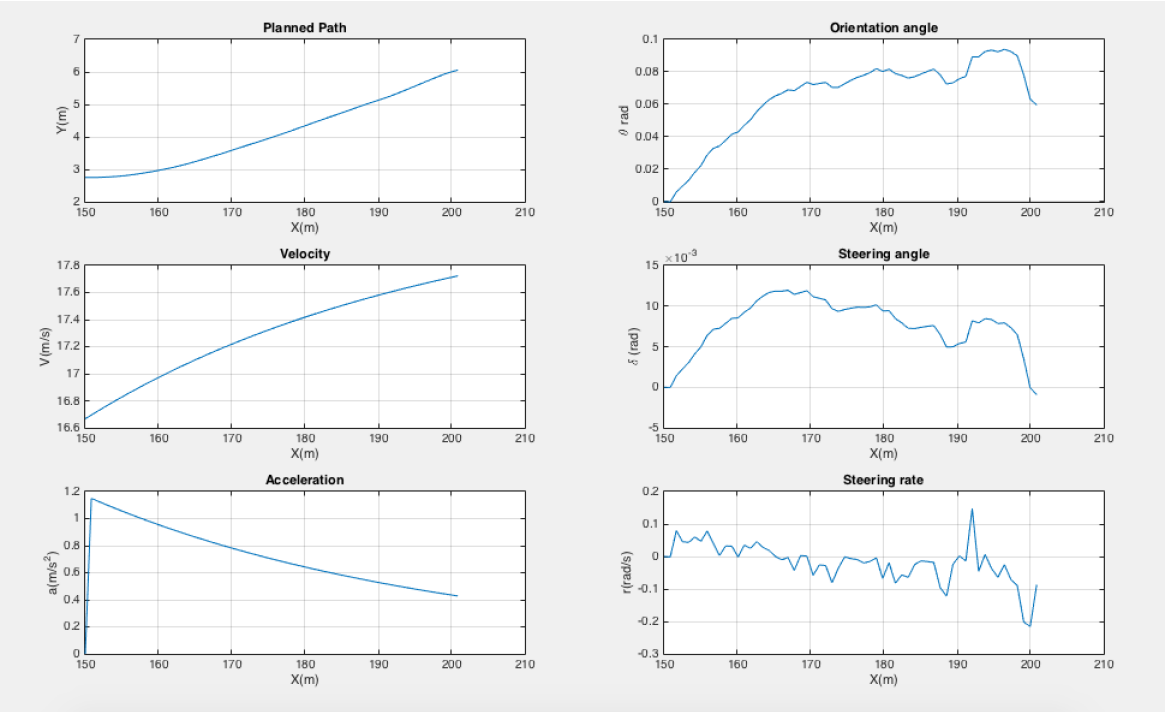


Figure 5-8: The generated trajectory. This trajectory is obtained with $V_{init} = 16.7m/s$ (ca. 60km/h).

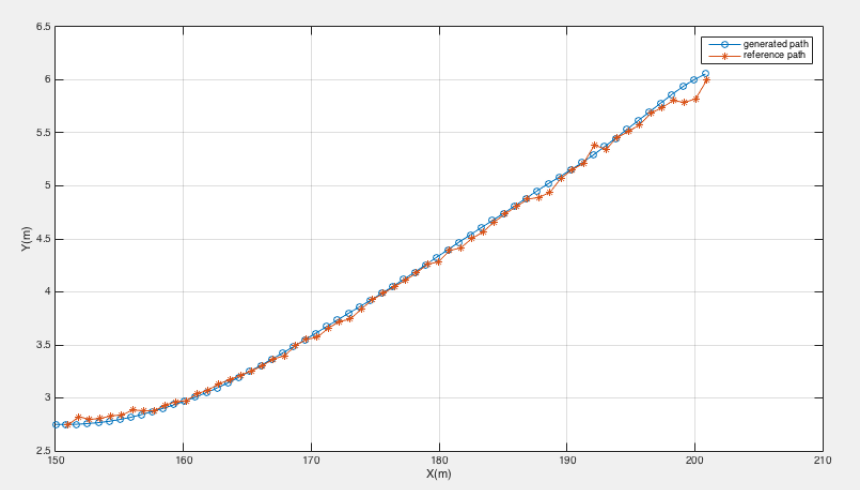


Figure 5-9: The reference path (the red line in this figure) and the corresponding generated path (the blue line in this figure).

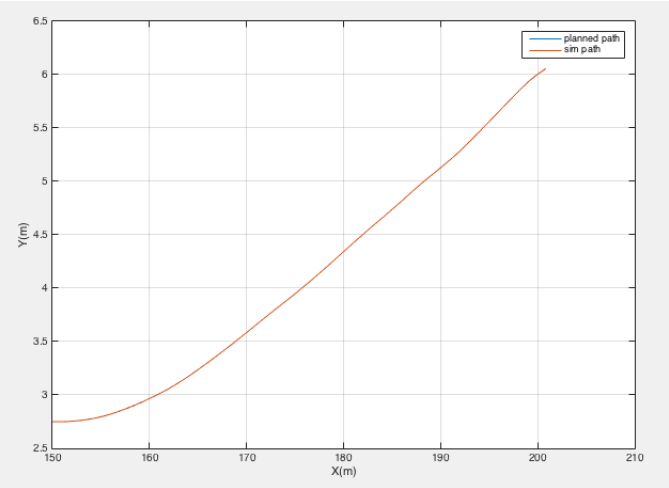


Figure 5-10: The planned trajectory (the orange line) and the reconstructed trajectory (the yellow line).

The stop criterion also affects the algorithm efficiency. In Figure 5-8, the orientation angle θ is 0.06 rad when the goal region is achieved. The stop criterion with respect to the θ is set to -0.1 $rad \leq \theta \leq 0.1$ rad in this case. We can also use a tighter criterion for θ. However, a tighter criterion means much efforts to achieve the goal. In this case, if we narrow the range to 0.05 $rad \leq \theta \leq 0.05$ rad, then no path can be found. The number of nodes needs to find the best safe path is more than 50000, and the time to generate 50000 nodes is larger than one thousand second.

To balance the computation time and the accuracy of the result, we propose to combine the criteria. That is, when the tree is far from the goal point, only two criteria are used to stop the algorithm. After the tree is close to the goal point, one or more criteria are added to stop the algorithm. The results are shown in Figure 5-11 and Figure 5-12. From Figure 5-11, it can be seen that the final orientation angle of the truck is less than 0.05 rad, and the lateral deviation is 0.02 m.

5-2-3 Discussion

In this section, we implement the RRT algorithm off-line. Two different vehicle systems are used in this section: the open-loop system and the closed-loop system. The open-loop system is the vehicle model without controller. The inputs for this system are the acceleration and the steering rate. The closed-loop system is the vehicle model with controller. The inputs for the closed-loop system are a set of reference segments.

Some parameters may affect the results of the algorithms. For the algorithm over the open-loop system, the integration step-size ΔT plays a significant role. A too small step-size leads to a huge computation time, whereas a too large step-size results in large deviations. For the algorithm over the closed-loop system, the stop criterion and the length of the step-size

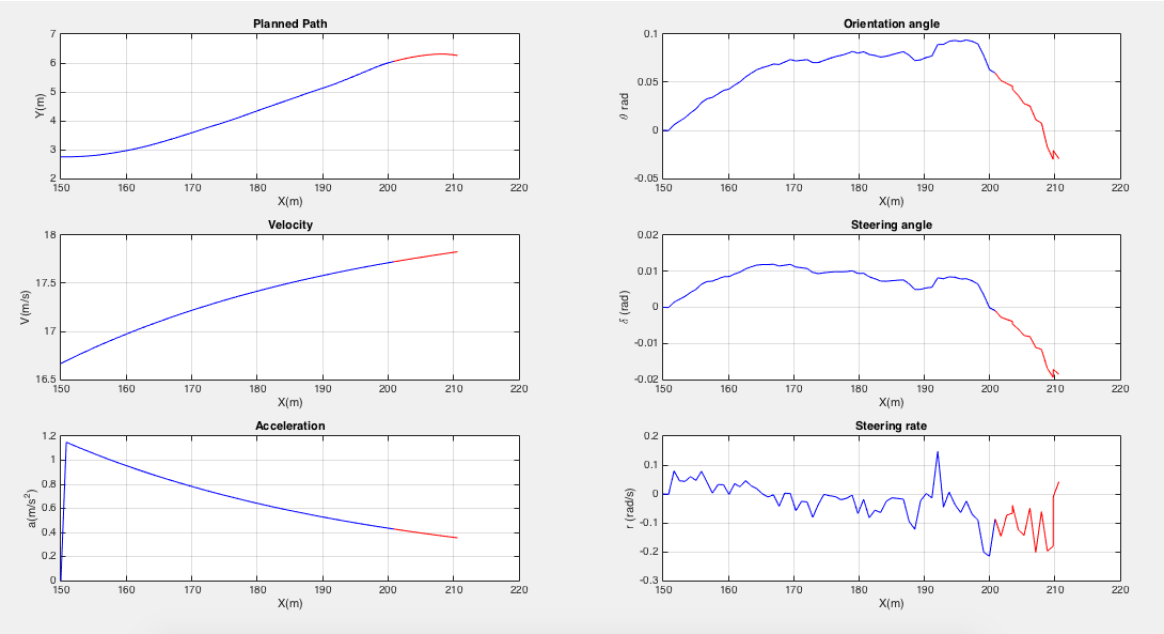


Figure 5-11: The generated trajectory. This trajectory is obtained with combined stop criteria.

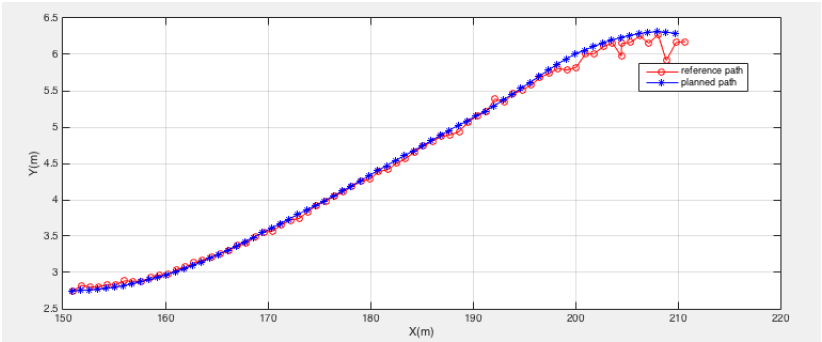


Figure 5-12: The reference path (the red line), and the planned trajectory (the blue line). This trajectory is obtained with combined stop criteria.

Δr impose obvious influence on the results. The length of the step-size Δr should chosen properly. A too large Δr may lead the algorithm failing to work, whereas a too small Δr will waste a lot of computation time. Based on the tuning experience, the value of Δr can be chosen as $\Delta r = 1\ m$ in this case.

As for the stop criterion, when we use too many criteria, the computation time is long. However, when we choose few criteria, the results are not good. To balance the computation time and the accuracy of the results, the combined criteria are proposed there. Comparing Figure 5-11 and Figure 5-8, we can see that the trajectories generated by combined stop criteria and single stop criterion show small difference on smoothness. But the trajectory with the combined stop criteria shows more accurate results. The final lateral deviation and the final orientation angle of the trajectory with the combined stop criteria are smaller than them of the trajectory with single stop criterion. Additionally, the computation time with the combined stop criteria is much lower than with the single stop criterion.

We have seen that there exist disturbances in the simulation results of open-loop system. In Figure 5-7 and Figure 5-6, large disturbances occur in the acceleration a and the steering rate r. This is because that the value of the acceleration a and the steering rate r that drive the vehicle from the current position to the next position are randomly sampled from the inputs space in the open-loop conditions. This disturbance can be reduced by sampling the rate of the acceleration $\dot{a}$ and the rate of the steering rate $\dot{r}$, instead of sampling acceleration a and steering rate r. This method has the potential to smooth the curves of the acceleration and steering rate, but it also increases the computation cost.

The disturbances appear in the simulation results of the closed-loop system as well. In Figure 5-10 and Figure 5-12, the reference path shows disturbance when it is near to the goal. This is because that the sampling probability is higher and the sampling range is narrower near the goal range than far from the goal range. It does not affect the performance of the closed-loop RRT algorithm, because the step size of the RRT algorithm is small and the reference path is only to provide the prorogation tendency of the algorithm. This phenomenon can be improved by reducing the maximum step size of the algorithm. In Figure 5-8 and Figure 5-11, the curves of the steering angle, orientation angle and the steering rate show different disturbance. In the closed-loop system, the steering angle is calculated by using the coordinates of the sampled nodes. Since the nodes are sampled randomly, the curve of the steering angle is not as smooth as of the velocity. Additionally, the values of the steering angle are small. The capacity of the computer itself is also contributed to the small disturbance. The disturbance of the steering rate seems to be large. This is because the steering rate is obtained by deviating the steering angle.

Besides, from Figure 5-6, Figure 5-7 and Figure 5-8, we can see that the trajectory planned by the closed-loop RRT algorithm shows less disturbance and is smoother than the trajectory planned by the open-loop RRT algorithm. Moreover, it is much difficult to limit the speed of the open-loop system by using RRT algorithm, because the inputs of the system are randomly sampled from the input space. The algorithm over the closed-loop dynamics gives a better performance than the algorithm over the open-loop dynamics. In the next section, we apply the closed-loop RRT algorithm to generate the trajectory in the real-time situation.

5-3 Real-Time Implementation

In this section, we apply the closed-loop RRT algorithm online. The closed-loop RRT algorithm uses a closed-loop model embedded with controllers to make the prediction. Actually, it is a method to generate control signals to the controller, and the lower-level system must use the same controller as the planner. Hence, we can validate the trajectory planned by the closed-loop RRT algorithm by sending the reference path sequences to the same controller, and check whether the planned path matches the reconstructed path by the same controller. The real-time implementation is based on the offline planning method.

The merging scenario of a truck from the ramp to the highway is described in Figure 5-13. We use the distance of CoGs of the PF vehicle and the PL vehicle to represent the length of the gap in the configuration space. Actually, the actual gap for the truck merging into is 5 m smaller than the gap since we consider the lengths of the surrounding vehicles are 5 m in the configuration space. The dynamics of the PF vehicle and the PL vehicle can be described by Equation 2-16. In different scenarios, the values of the accelerations and gap are different. The scenarios are described in Table 5-7.

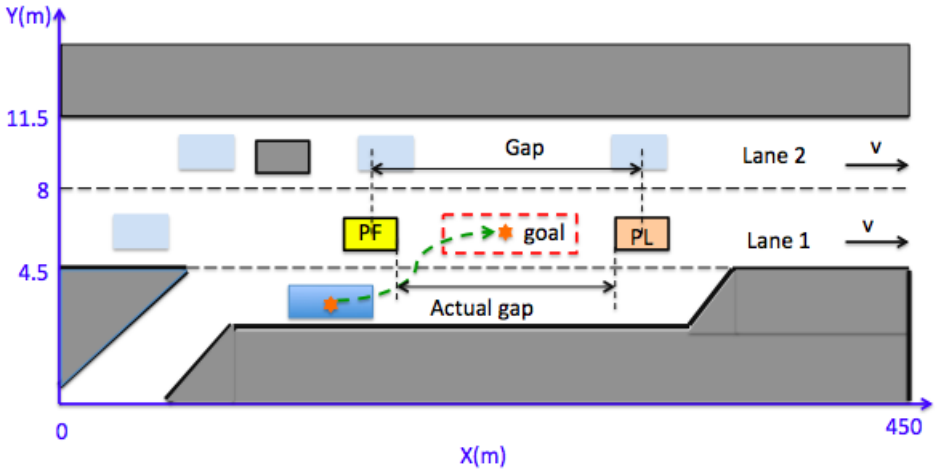


Figure 5-13: The merging scenario of a truck from the ramp to the highway in the real-time implementation. The initial speed of the truck is 16.7 m/s (ca.60 km/h). The speed of the vehicles in the target lane is 18.3 m/s.

Scenario	Gap	PL acceleration	PF acceleration	Start point	Goal point
6	100 m	0	0	[130,2.75,0]	[210,6.25,0]
7	50 m	0	0	[120,2.75,0]	[210,6.25,0]
8	50 m	0	0.5 (m/s^2)	[145,2.75,0]	[200,6.25,0]
9	50 m	-0.5 (m/s^2)	0.	[165,2.75,0]	[220,6.25,0]
10	40 m	0	0	[155,2.75,0]	[210,6.25,0]

Table 5-7: Different scenarios used in the real-time implementation.

5-3-1 Real-Time Implementation in Different Scenarios

Firstly, we consider a large gap to execute the merging manoeuvre. Consider the scenario 6 that a truck is moving on the ramp of the highway, and intends to merge into the target lane. The vehicles move in the target lane with the speed $V = 18.3\ m/s$ (ca. 66 km/h) and acceleration $a = 0\ m/s^2$. The gap between the putative leader vehicle (PL) and putative follower vehicle (PF) is 100 m. We update the environment every $\Delta t = 0.05$ s. The tolerance of the goal is $[50\ m, 0.2\ m]$.

Figure 5-14 shows the generated trajectory as well as its corresponding heading angle, velocity, acceleration, steering angle and steering rate. In Figure 5-14, the coordinates is [198 m,6.05 m], the orientation angle is $\theta = 0.058\ rad$ and the steering angle is $\delta = 0.001\ rad$ at the final state. Although the deviation of the orientation angle and the steering angle are small, the deviation in Y-axis is not so small.

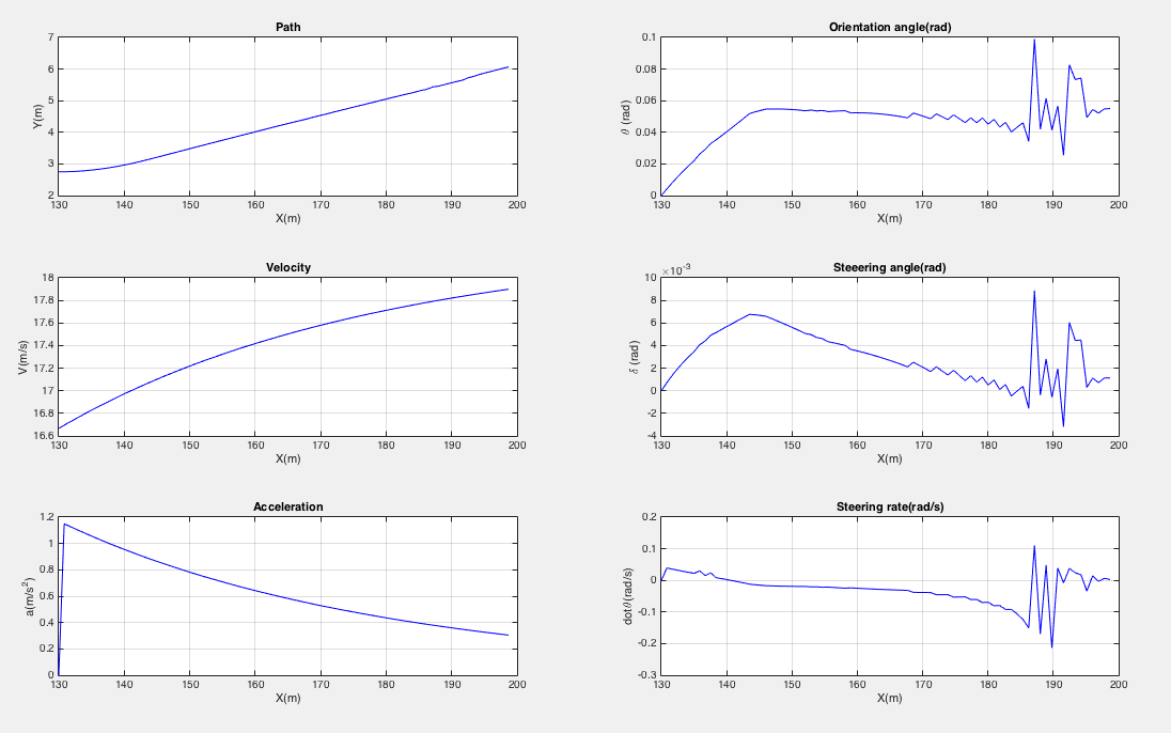


Figure 5-14: The planned path, orientation angle, velocity, steering angle, acceleration and steering rate. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05$ s, $gap = 100$ m, $a = 0\ m/s^2$, and truck initial speed is $V = 16.7\ m/s$ (ca. 60 km/h).

In Figure 5-14,the steering angle and the steering rate show slight disturbances near the goal point. This is caused by the chaotic behaviour of the reference path near the goal region. The sampling probability increases near the goal region, but the sampling time at each planning cycle is limited. If we can prolong the sampling duration, this means we can have more choices for the best safe node sequence, and then a better trajectory can be generated. However, it is not a good method to prolong the sampling duration, since it will largely increase the computation burden. According to our own experience, at most times, even though we increase

the sampling duration, the disturbances cannot be eliminated.

To obtain better trajectory, we apply the combined stop algorithm on the online implementation. The criteria used when the tree is near to the goal point are more than used when the tree is far from the goal point. In the previous chapter, we have discussed that if we use too many criteria to stop the algorithm at the very beginning, the algorithm may fail to work because of the huge computational burden. However, when the truck is located near the goal point, adding more criteria to stop the algorithm really helps to find better result with relative lower computational cost.

Taking scenario 6 as example, after the truck approaches near to the goal point, the criteria would be set over both the goal region and the orientation angle. The algorithm will stop if and only if all the criteria are satisfied. In the scenario 6, we set the deviation in the lateral direction as not more than 0.2 m when the truck is far from the goal point. However, the lateral limitation would be set as less than 0.1 m when the truck is close to the goal point. Additionally, the criterion over the orientation angle θ is introduced as -0.02 $rad < \theta < 0.02$ rad here. Now, the tolerance of the goal becomes [20 m, 0.1 m, 0.02 rad].

The Figure 5-15 presents the results with the combined criteria. The Figure 5-17 shows the central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle, respectively. The coordinates is [210 m,6.28 m], the orientation angle is $\theta = -0.016$ rad and the steering angle is $\delta = -0.008$ rad at the final state inn this case.

In the scenario 7, we choose a small gap with respect to scenario 6. Consider the truck is moving on the ramp of the highway, and intends to merge into the target lane. The vehicles move in the target lane with the speed 18.3 m/s (ca.$V = 66$ km/h) and $a = 0$. The gap between the putative leader vehicle (PL) and putative follower vehicle (PF) is 50 m. The actual gap is 45 m. We update the environment every $\Delta t = 0.05$ s. The tolerances of the goal are [40 m, 0.2 m] and [20 m, 0.1 m, 0.02 rad].

Figure 5-18 shows the generated trajectory. Figure 5-19 compares the reference path, reconstructed path and the planned path. Figure 5-20 shows the central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle, respectively, in this scenario. The coordinates is [210 m,6.27 m], the orientation angle is $\theta = -0.019$ rad and the steering angle is $\delta = -0.007$ rad at the final state.

Then, we give acceleration to the PF vehicle in the target lane. Consider the scenario 8 that a truck is moving on the ramp of the highway, and intends to merge into the target lane. The vehicles move in the target lane with the speed $V = 18.3$ m/s (ca.66 km/h). At this time, the PF vehicle intends to accelerate with $a_{PF} = 0.5$ m/s^2. The gap between the putative leader vehicle (PL) and putative follower vehicle (PF) is 50 m. We update the environment every $\Delta t = 0.05$ s. The tolerances of the goal are [40 m, 0.2 m] and [20 m, 0.1 m, 0.02 rad].

Figure 5-21 shows the generated trajectory. Figure 5-22 compares the reference path, reconstructed path and the planned path. Figure 5-23 shows the central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower

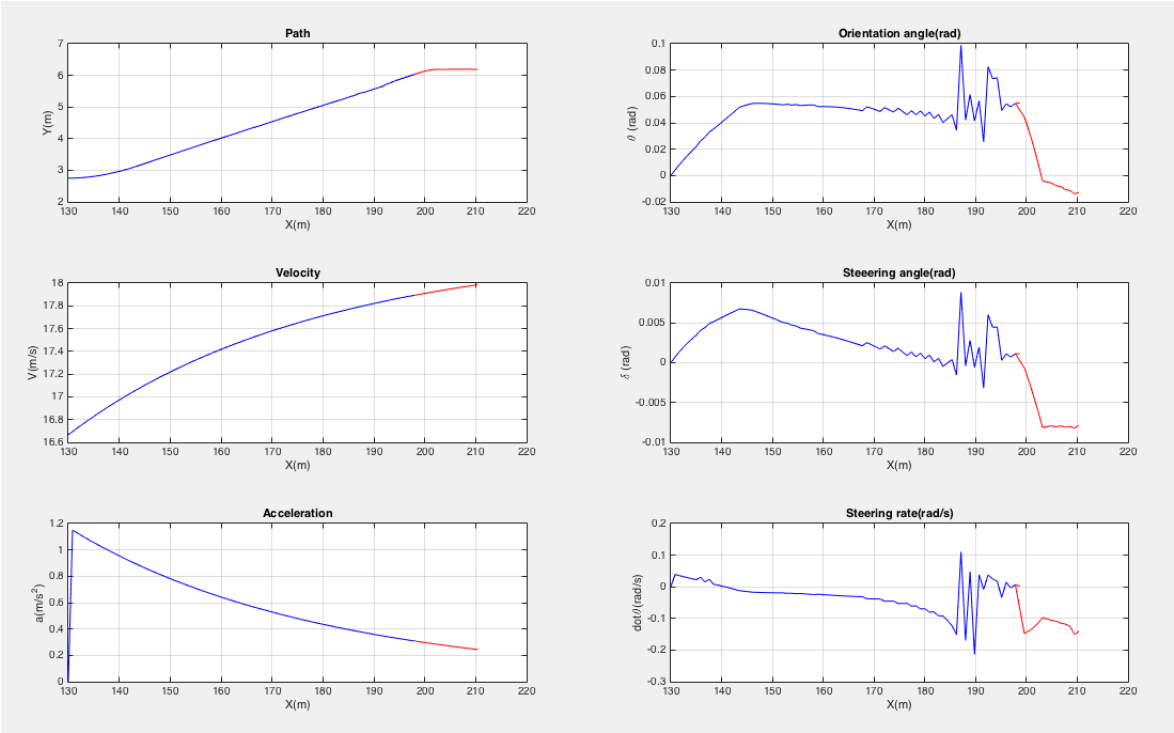


Figure 5-15: The planned path, orientation angle, velocity, steering angle, acceleration and steering rate. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05\ s$, $gap = 100\ m$, $a = 0\ m/s^2$, and truck initial speed is $V = 16.7\ m/s$ (ca.60 km/h).

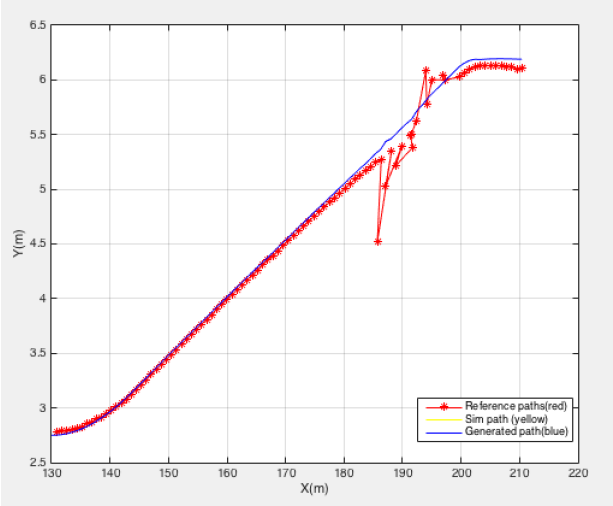


Figure 5-16: The planned path, reference paths and reconstructed path. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05\ s$, $gap = 100\ m$, $a = 0\ m/s^2$, and truck initial speed is $V = 16.7\ m/s$ (ca.60 km/h).

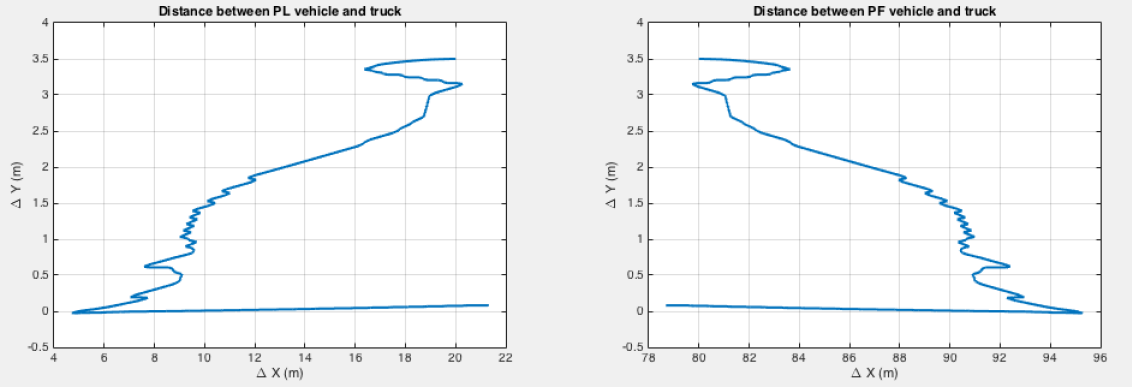


Figure 5-17: The central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle, respectively. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a = 0\ m/s^2$, $\Delta t = 0.05\ s$, $gap = 100\ m$, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h).

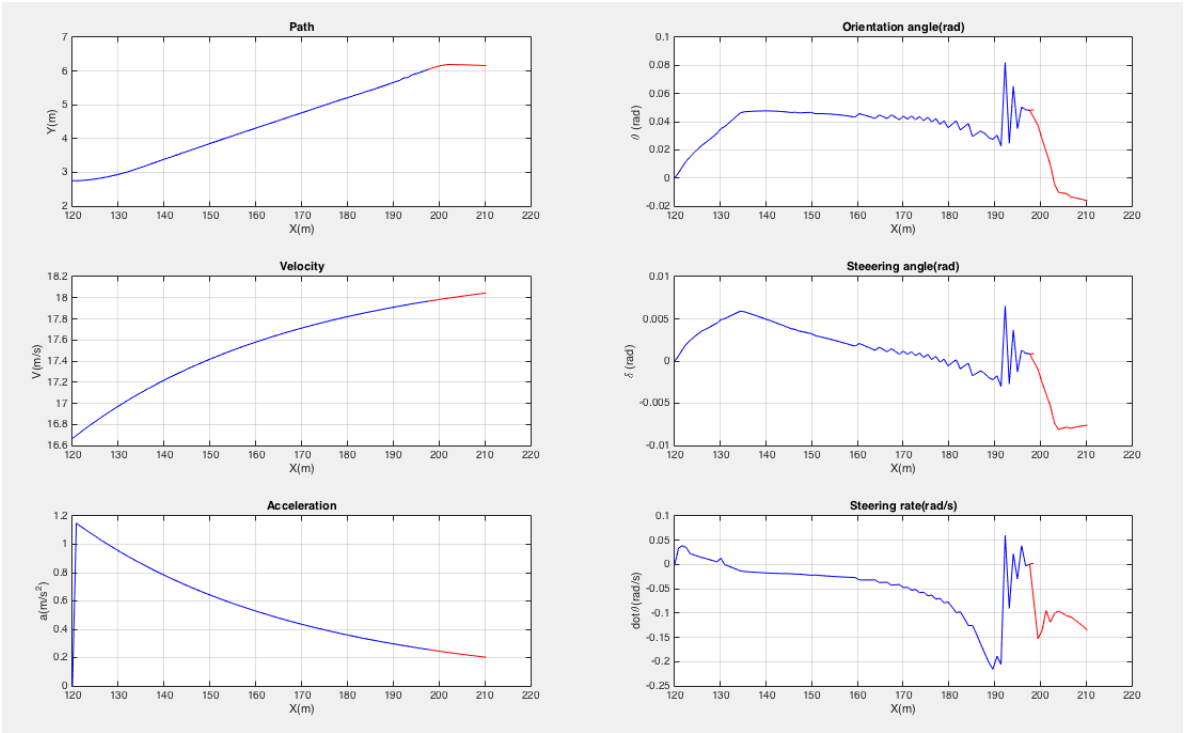


Figure 5-18: The planned path, orientation angle, velocity, steering angle, acceleration and steering rate. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05$ s, $gap = 50$ m, $a = 0$, and truck initial speed is $V = 16.7\ m/s$ (ca.60 km/h).

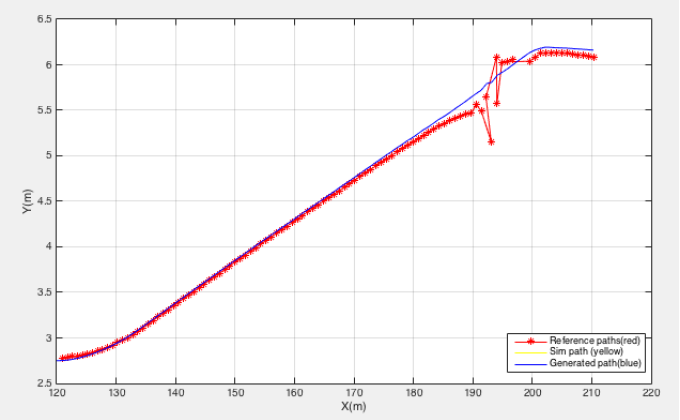


Figure 5-19: The planned path, reconstructed path and the reference path. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a = 0$, $\Delta t = 0.05$ s, $gap = 50$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h).

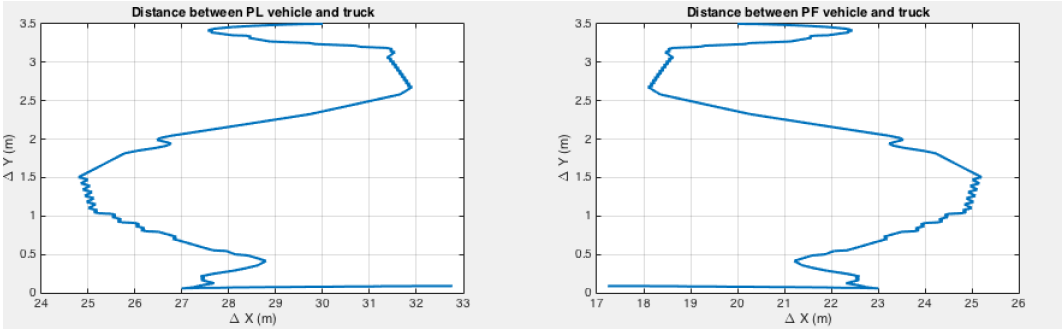


Figure 5-20: The central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle, respectively. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a = 0$, $\Delta t = 0.05$ s, $gap = 50$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h).

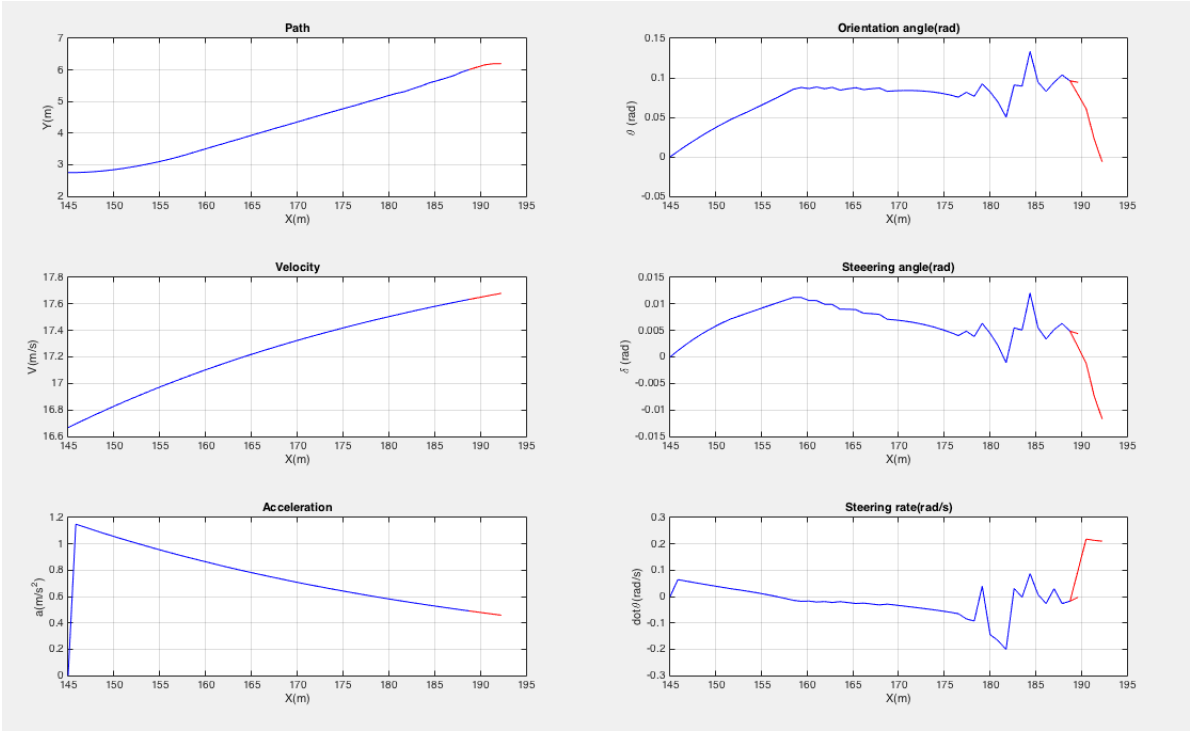


Figure 5-21: The planned path, orientation angle, velocity, steering angle, acceleration and steering rate. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05$ s, $gap = 50$ m, $a_{PF} = 0.5\ m/s^2$, and truck initial speed is $V = 16.7\ m/s$ (ca.60 km/h).

vehicle in x-axis and y-axis, respectively, in this scenario. The coordinates is [192 m,6.28 m], the orientation angle is $\theta = -0.002\ rad$ and the steering angle is $\delta = -0.012\ rad$ at the final state.

Besides, we can give deceleration to the PL vehicle in the target lane. Consider the scenario 9 that a truck is moving on the ramp of the highway, and intends to merge into the target lane. The vehicles move in the target lane with the speed 18.3 m/s (ca.$V = 66km/h$). At this time, the PL vehicle intends to decelerate with $a_{PL} = -0.5\ m/s^2$. The gap between the putative leader vehicle (PL) and putative follower vehicle (PF) is 50 m. We update the environment every $\Delta t = 0.05\ s$. The tolerances of the goal are [40 m, 0.2 m] and [20 m, 0.1 m, 0.02 rad].

Figure 5-24 shows the generated trajectory. Figure 5-25 compares the reference path, reconstructed path and the planned path. Figure 5-26 shows the central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle in x-axis and y-axis, respectively, in this scenario. The coordinates is [218 m,6.23 m], the orientation angle is $\theta = -0.019\ rad$ and the steering angle is $\delta = -0.012\ rad$ at the final state.

Lastly, we try to reduce the gap more and set limitations to the actual gaps among the PF vehicle, the truck and the PL vehicle. Consider the scenario 10 that a truck is moving on the

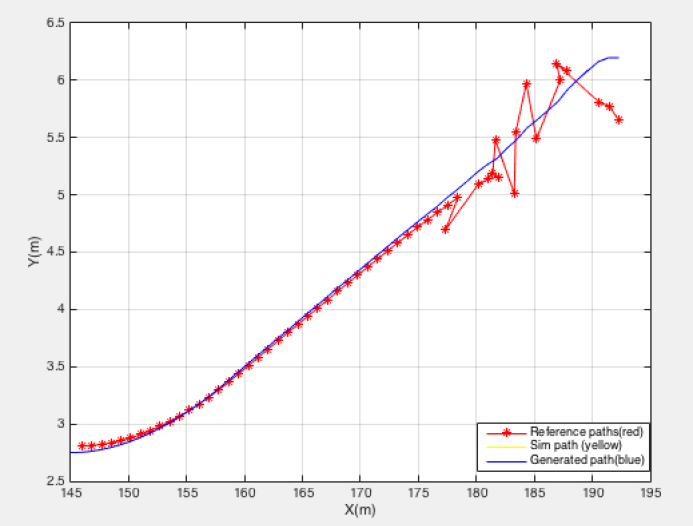


Figure 5-22: The planned path, reconstructed path and the reference path. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a_{PF} = 0.5\ m/s^2$, $\Delta t = 0.05$ s, $gap = 50$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h).

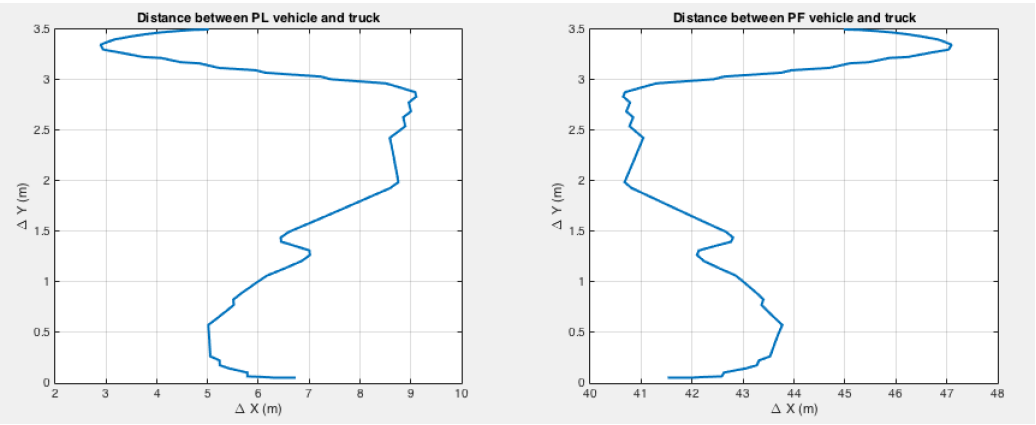


Figure 5-23: The central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle, respectively. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a_{PF} = 0.5\ m/s^2$, $\Delta t = 0.05$ s, $gap = 50$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h).

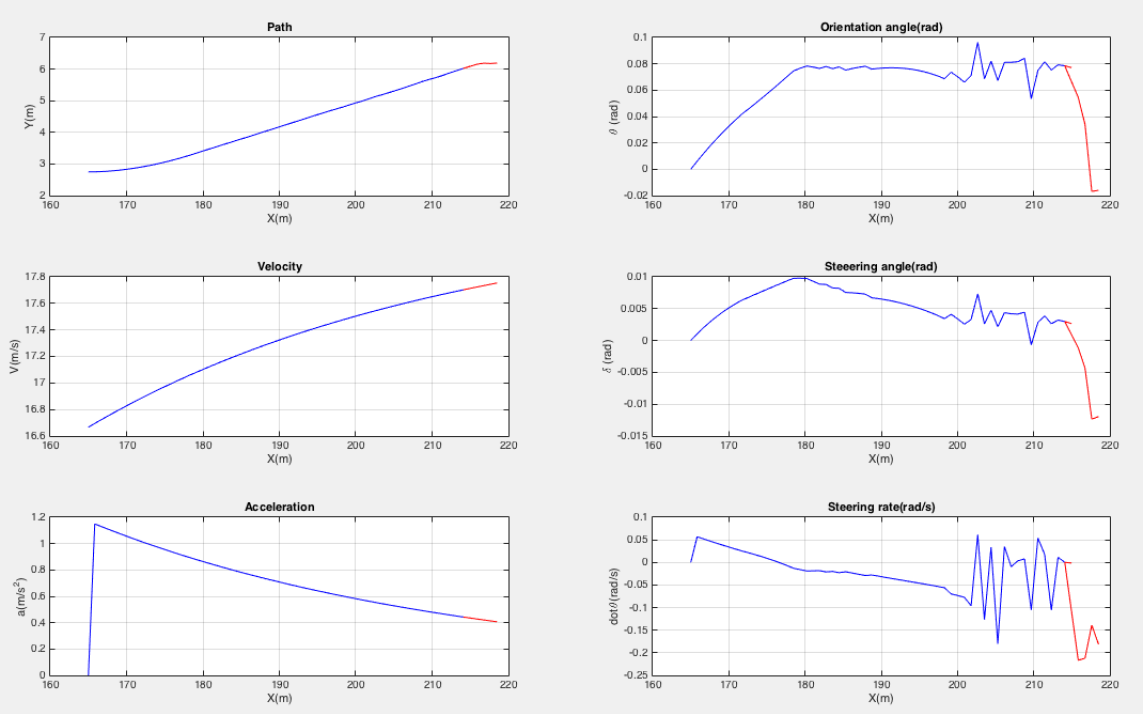


Figure 5-24: The planned path, orientation angle, velocity, steering angle, acceleration and steering rate. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05$ s, $gap = 50$ m, $a_{PL} = -0.5\ m/s^2$, and truck initial speed is $V = 16.7\ m/s$ (ca.60 km/h).

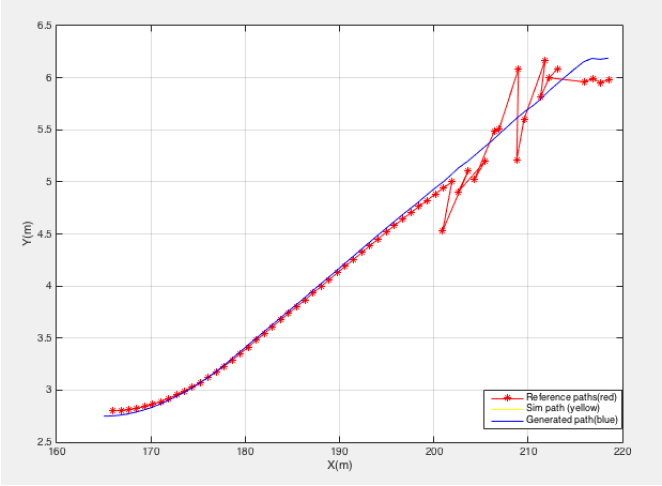


Figure 5-25: The planned path, reconstructed path and the reference path. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a_{PL} = -0.5\ m/s^2$, $\Delta t = 0.05$ s, $gap = 50$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h).

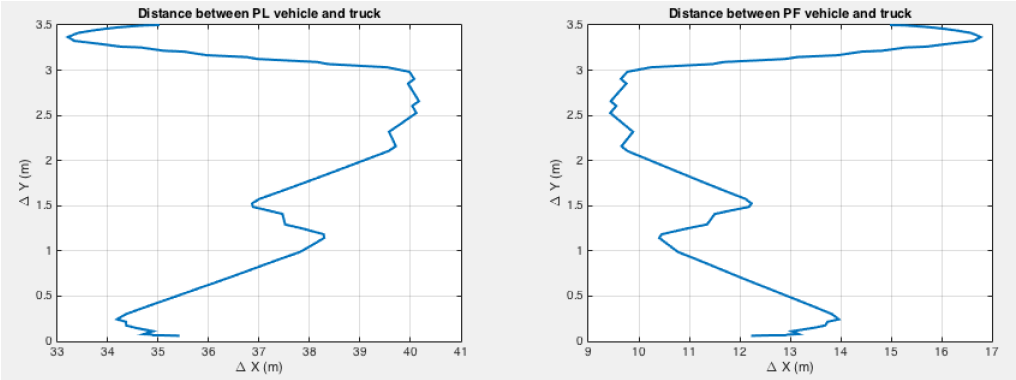


Figure 5-26: The central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle, respectively. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a_{PL} = -0.5\ m/s^2$, $\Delta t = 0.05$ s, $gap = 50$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h).

ramp of the highway, and intends to merge into the target lane. The vehicles move in the target lane with the speed 18.3 m/s (ca.$V = 66km/h$) and $a = 0\ m/s^2$. The gap between the putative leader vehicle (PL) and putative follower vehicle (PF) is reduced to $40m$. We update the environment every $\Delta t = 0.05\ s$. The tolerances of the goal are $[40\ m, 0.2\ m]$ and $[20\ m, 0.1\ m, 0.02\ rad]$. In order to guarantee the safety after the truck merges into the target lane, we set a limitation to the gaps between two vehicles at the final state. We assume the acceptable gap as 1 s, then the distances between the PF vehicle and the truck, and between the truck and the PL vehicle must be larger than 9.15 m, since the velocities of the PF and PL vehicle are 18.3 m/s.

Figure 5-27 shows the generated trajectory. Figure 5-28 compares the reference path, reconstructed path and the planned path. Figure 5-29 shows the central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle, respectively, in the scenario 11. The coordinates is [202 m,6.27 m], the orientation angle is $\theta = 0.03\ rad$ and the steering angle is $\delta = -0.006\ rad$ at the final state. The actual gap between the PF vehicle and truck is 16.4 m, and between the truck and PF vehicle is 9.6 m.

5-3-2 Discussion

In this section, we implement the real-time algorithm in different scenarios and show the trajectories generated by the online algorithm in these scenarios. The results are concluded in Table 5-8.

In the scenario 6, the gap between the PF vehicle and the PL vehicle is 100 m and it keeps constant during all the manoeuvres. We give the same acceleration and initial speed to the PF vehicle and the PL vehicle, which does not affect the length of the gap, but it affects the velocities of PF vehicle and PL vehicle. Based on the results, we can see that the algorithm can find a smooth trajectory for the truck. Although there are some disturbances of the angles

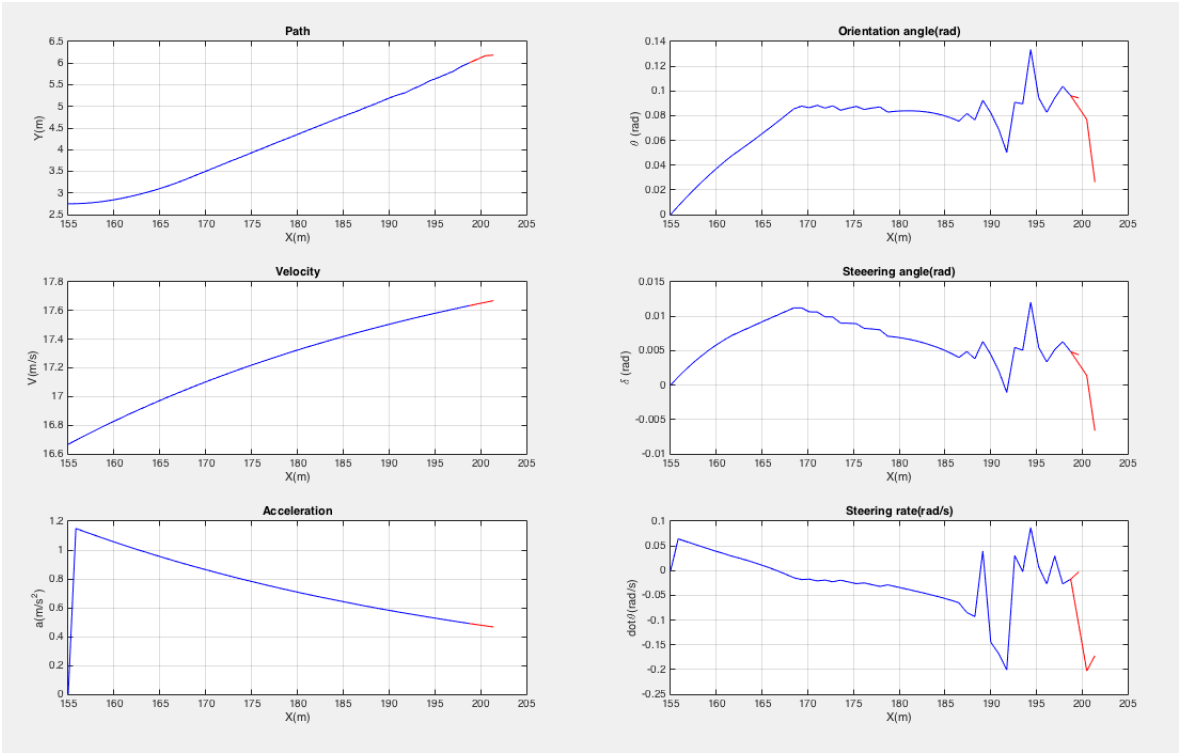


Figure 5-27: The planned path, orientation angle, velocity, steering angle, acceleration and steering rate. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $\Delta t = 0.05\ s$, $gap = 40\ m$, $a = 0\ m/s^2$, and truck initial speed is $V = 16.7\ m/s$ (ca.60 km/h).

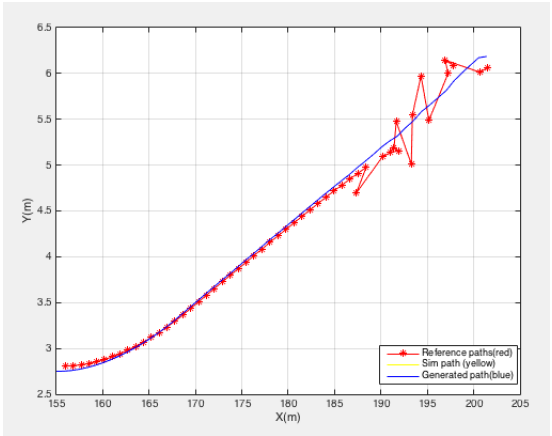


Figure 5-28: The planned path, reconstructed path and the reference path. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a = 0\ m/s^2$, $\Delta t = 0.05$ s, $gap = 40$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h).

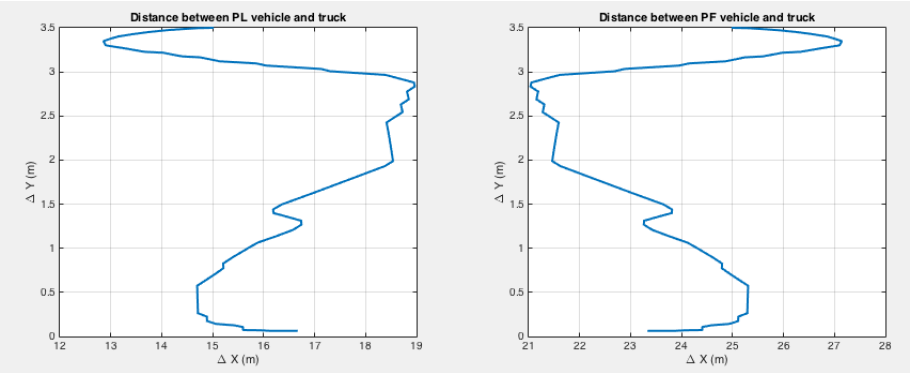


Figure 5-29: The central distance of the truck and the putative leader vehicle, and the central distance of the truck and the putative follower vehicle in x-axis and y-axis, respectively. This figure is obtained by setting obstacle speed $V_{ob} = 18.3\ m/s$ (ca.66 km/h), $a = 0\ m/s^2$, $\Delta t = 0.05$ s, $gap = 40$ m, and truck initial speed $V = 16.7\ m/s$ (ca.60 km/h).

Scenario	Goal state	Orientation deviation	Lateral deviation	Related Figures
6	[210,6.25,0]	0.016 rad	0.03 m	Figure 5-16,Figure 5-15,Figure 5-17
7	[210,6.25,0]	0.019 rad	0.02 m	Figure 5-18,Figure 5-19,Figure 5-20
8	[200,6.25,0]	0.002 rad	0.03 m	Figure 5-21,Figure 5-22,Figure 5-23
9	[220,6.25,0]	0.019 rad	0.02 m	Figure 5-24,Figure 5-25,Figure 5-26
10	[210,6.25,0]	0.020 rad	0.02 m	Figure 5-27,Figure 5-28,Figure 5-29

Table 5-8: Results of different scenarios in this section. The state includes the coordinates of the trucks CoG and the orientation angle in this table.

and the steering rate occurring at the end of the trajectory, these disturbances are very small.

In this section, we also compare the results obtained by the single stop criterion and the combined stop criteria. Based on the simulation results, it can be seen that the results obtained with the combined stop criteria are better than the results obtained with the single stop criterion. The trajectory generated by combining different stop criteria has smaller lateral deviation with regard to the goal point, and can adjust the orientation angle to a smaller value when the truck has merged into the gap.

In the scenario 7, we apply the online RTT algorithm with the constant gaps that are smaller than the constant gap of first scenario. Based on these results, we can see that the algorithm succeeds to plan the trajectories when the gaps is 50 m. In the 8th and 9th scenario, we implement the online algorithm on the changing gap. We give small acceleration and deceleration to the PF vehicle and PL vehicle, respectively. That makes the gap becoming smaller and smaller during the merging maneuvers. In these two scenarios, the online algorithm also succeeds to find the smooth trajectories for the truck. In the scenario 10, we reduce the size of the gap to 40 m and set a limitation of the distance among the PF vehicle, the truck and the PL vehicle in order to guarantee the safety after the truck merges into the target lane. The simulation results indicates that the actual gaps among the PF vehicle, the truck and the PL vehicle are larger than 9.15 m.

5-4 Summary

In this chapter, we determine the bias for the sampling strategy, and represent the simulation results of the offline algorithm and online algorithm for merging maneuvers. Firstly, the bias of the sampling space for merging maneuvers is determined. In the middle of this chapter, the offline and online implementation results are depicted.

The bias for the sampling strategy is determined offline in the first section of this chapter. The results show that the algorithm with bias $\sigma_r = 3\ m$ and $\sigma_\theta = \frac{\pi}{8}\ rad$ is most suitable for this work. Moreover, the bias helps to improve the efficiency of the algorithm. The bias parameterised in this section is used for the offline and online implementation in this work.

The offline algorithm is applied on both the open-loop system and the closed-loop system. In this part, we compare the parameters of the algorithm that affect the results, and the trajectories planned by single stop criterion and the combined stop criteria. By using the combined stop criteria, we can obtain more accurate results with less computational cost. Besides, we also compare the results obtained with the open-loop system and the closed-loop system.

The online algorithm is a combination of the offline algorithm and the replanning algorithm. In this chapter, we apply the online algorithm in five different scenarios, and compare the results obtained by the single stop criterion and the combined stop criterion in the first scenario. The five different scenarios have different sizes of gap and dynamics of the PF and PL vehicle. The online algorithm can generate smooth trajectories in these scenarios.

Chapter 6

Conclusions and Recommendations for Future Work

This thesis presents the real-time implementation of the Rapidly-exploring Random Tree (RRT) algorithm for planning the trajectory for a truck on the highway. Both the open-loop RRT algorithm and the closed-loop RRT algorithm are applied to plan a trajectory to the truck in the offline planning. The replanning algorithm is applied on the closed-loop system. Simulations of the real-time implementations are done with different scenarios.

The main contributions of this thesis are:

- The proposal of the online RRT algorithm for the trajectory generation for trucks in merging manoeuvres on the highway.
- The proposal of the ideas to improve the efficiency of the method.
- The implementation of the method in MATLAB/SIMULINK and analysis.

Additionally, the implementation of the presented algorithms in MATLAB code will allow for future research of automated vehicles in the environment of MATLAB/SIMULINK.

6-1 Discussions and Conclusions

In this thesis, we develop the real-time RRT algorithm for a truck in merging manoeuvres on the highways, and apply the off-line RRT algorithm on both the open-loop system and the closed-loop system. The RRT algorithm is a flexible algorithm. Every parameter makes a contribution to the final result. We have discussed the effects of the step size on the open-loop RRT algorithm.

We use several strategies to improve the efficiency of the RRT algorithm in this thesis. The first one is to change the sampling probability of the points in the configuration space, and a suitable bias is chosen for a truck in the merging manoeuvres. Additionally, we propose a rule helps to find the bias. The second strategy is to make more trials for node selection. The last strategy is to use combined criteria to stop the propagation of the tree.

With regard to the collision detection, the bounding volume is introduced to model the truck and the obstacles on the highway. We discuss the different bounding volumes for collision detection, and finally we use the OBB for the truck. To improve the safety, we enlarge the lateral sizes of the truck and the surrounding vehicles when detecting the collision. Considering the potential errors caused by the inertia of vehicles and the latency of the system, a incremental part is attached to the vehicles when checking collisions.

The main conclusions that are made in this thesis work, with regard to the real-time implementation of a sample-based trajectory generator for trucks in merging manoeuvres on highways, are:

- The results of the offline implementation over the open-loop RRT algorithm and closed-loop RRT algorithm show that the closed-loop RRT algorithm can plan a smoother trajectory than the open-loop RRT algorithm.
- The step size of the RRT algorithm makes a contribution to the results. The results show that a small step size means a smooth trajectory, as well as more computational effort. A suitable step size can improve the performance of the RRT algorithm.
- The combined stop criteria can improve the efficiency of the algorithm. This strategy uses different criteria when the vehicle is far from the goal and when it is close to the goal. When the vehicle is far from the goal, we focus mainly on the changes of the location and give more tolerance to the orientation angle. However, when the vehicle is close to the goal, the tolerance of the orientation angle becomes small.
- The sample bias can be inserted into the sampling function. The simulation results illustrate that the bias indeed improves the efficiency of the RRT algorithm in path planning.

6-2 Recommended Future Work

1. **Collision Detection Method** This thesis uses the line-line intersection method to detect the collision. It is the simplest method to detect the collision, and only works well with simple regular polygons. Future work can focus on a new efficient collision detection method.
2. **Controllers** A PID controller and a pure-pursuit controller are used to control the speed and the steering, respectively. In this work, we develop the real-time algorithm over the closed-loop system, and the outputs of the trajectory generator are a set of segments. Usually, we use the same controllers for the vehicle to track these segments.

Future work can try to use other different controllers with a more complex vehicle model to track these segments. Besides, we use 2nd Ziegler-Nichols to tune the system in this work, but it does not work well. Future works can focus on the other method to tune the PID controller.

3. **Code in C/C++** The code used in the simulation is written with MATLAB. To speed up the computation time, we can transfer the MATLAB code into C/C++ in the future.

4. **Implementation in Other Fields** The RRT algorithm is one of the solutions for the path-planning problems. In this work, we implement it for the truck's merging manoeuvres on the highway. Future works can be focused on applying the RRT algorithm on the other fields. For example, the RRT algorithm for solving the path-planning problem of 3D printer and the grasp robot arms that have more complex dynamic models, for planning the cutting route for the CNC machine, for planning the path for the unmanned detective helicopter, for planning the path of the rescue robot in dangerous area.

5. **Comparison with other methods** This work mainly focus on RRT algorithm applying on the trajectory generation problem. The future work can focus on the quantitative comparisons between the RRT algorithm and other trajectory generation methods, such as the multi-query method, optimisation-based method, spline-based method and so on.

Appendix A

Appendix

A-1 MATLAB Code

A-1-1 Offline RRT Algorithm over Open-Loop System

```
clear all;
clc;

max_iter=100000;
max_nodes=max_iter;
rand_seed=1;
map=struct('name', 'obstacle23.mat', 'start_point', [180,2.75,0], '
    goal_point', [230,6.25,0]);
variant='RRT_timestop_ol';



conf = struct;
%conf.delta_goal_point = 1;           % Radius of goal point
conf.delta_near = 0.5;               % Radius for neighboring nodes
conf.max_step = 1; % Maximum position change when we add a new node to
    the tree
conf.W=3; % width/2 of the vehicle
conf.L=9; % length/2 of the vehicle
conf.U_boundary=[-2.5;1.5;-0.1;0.1]; %[a,r]simple input constraints
conf.delta_goal_point =[10,0.15];
dist=0;
ind=0;
% plot traffic lanes
XYline=struct;
XYline.x1=0:0.1:460;
XYline.y1=0*XYline.x1+8;
XYline.x2=100:0.1:355;
```

```
XYline.y2=0*XYline.x2+4.5;


% using 'CRRT'

rrtree= eval([variant '(rand_seed, max_nodes, map, conf);']);


%%% Starting a timer
tic;

%%%%%%%%%
%for ind = 1:max_iter
while (dist==0)
    ind=ind+1;
    rrtree.update_ob(0,0)
    new_node = rrtree.sample_XY();%5*1

    [nearest_node,nearest_index] = rrtree.nearest(new_node);

    x_compare=zeros(8,5); %[x;y;theta;v,delta]
    x_init=nearest_node;

    for j=1:5
    input=rrtree.sample_U();
     if x_init(4)>66/3.6
         input(1)=0;
     end

    x_end=rungakutta(x_init,input);%8*1

    x_compare(:,j)=x_end;

    end

    new_node=selectnode(x_compare,[210;6.25;0]);

    new_node=rrtree.steer_off(nearest_index,new_node);



    if(~rrtree.obstacle_collision(new_node))
        rrtree.insert_node(nearest_index, new_node);
        dist=rrtree.checkgoal_offline(new_node);
    end

     if(mod(ind, 10) == 0)
         disp([num2str(rrtree.nodes_added-1) ' nodes in ' num2str(toc)]);
     end
end
```

```
80
81
82 % end timer
83 toc;
84
85 % plot
86 figure(1)
87 rrtree.findpath();
88 rrtree.plotpath();
89 hold on
90 p2=rrtree.plotobstacle();
91 hold on
92 plot(XYline.x1,XYline.y1,'black--', 'LineWidth',2);
93 hold on
94 plot(XYline.x2,XYline.y2,'black--', 'LineWidth',2);
95
96 figure(2)
97 subplot(3,2,1)
98 plot(rrtree.path(1,:),rrtree.path(2,:));
99 xlabel('x(m)')
100 ylabel('y(m)')
101 title('Planned path')
102 subplot(3,2,2)
103 plot(rrtree.path(1,:),rrtree.path(3,:));
104 xlabel('x(m)')
105 ylabel('\theta (rad)')
106 title('Orientation angle')
107 subplot(3,2,3)
108 plot(rrtree.path(1,:),rrtree.path(4,:));
109 xlabel('x(m)')
110 ylabel('v(m/s)')
111 title('Velocity')
112 subplot(3,2,4)
113 plot(rrtree.path(1,:),rrtree.path(5,:));
114 xlabel('x(m)')
115 ylabel('\delta (rad)')
116 title('Steering angle (rad)')
117 subplot(3,2,5)
118 plot(rrtree.path(1,:),rrtree.path(6,:));
119 xlabel('x')
120 ylabel('a(m/s2)')
121 title('Acceleration(m/s2)')
122 subplot(3,2,6)
123 plot(rrtree.path(1,:),rrtree.path(7,:));
124 xlabel('x(m)')
125 ylabel('r(rad/s)')
126 ylabel('Steering rate (rad/s)')
```

A-1-2 Offline RRT Algorithm over Closed-Loop System

```
clear all;
clc;

max_iter=50000;
max_nodes=max_iter;
rand_seed=1;
ind=0;
dist=0;
map=struct('name', 'obstacle20.mat', 'start_point', [150,2.75,0], '
    goal_point', [200,6.25,0]);
variant='RRT_timestop';


conf = struct;
conf.delta_goal_point = [50,0.4];          % Radius of goal point
conf.delta_near = 2;               % Radius for neighboring nodes
conf.max_step = 1; % Maximum position change when we add a new node to
    the tree
conf.W=3; % width/2 of the vehicle
conf.L=7; % length/2 of the vehicle
conf.U_boundary=[-2.17;1.5;-0.1;0.1];




% plot traffic lanes
XYline=struct;
XYline.x1=0:0.1:460;
XYline.y1=0*XYline.x1+8;
XYline.x2=100:0.1:355;
XYline.y2=0*XYline.x2+4.5;

%using class
rrtree= eval([variant '(rand_seed, max_nodes, map, conf);']);


%%% Starting a timer
tic;

%%%%%%%%%

while (dist==0)
    rrtree.update_ob(0);
    new_node = rrtree.sample_XY(); %3*1 sample configuration space
    u=rrtree.sample_U(); %2*1 sample inputs

    [nearest_node,nearest_index] = rrtree.nearest(new_node); %[x,y,theta,
        v] find nearest node in the tree from the sampled node


     x_init=nearest_node;
     u1=u(1,1);
     u2=u(2,1);
```

```
51        simout=sim('V_model','AbsTol','1e-4');
52        new=simout.get('simout');
53        Anode=new.Data';
54        node=Anode(:,end); % pick up the 10th node[x,y,theta,v,steeringangle
              ,acc]
55        new_node=[node;u];
56
57       if(~rrtree.obstacle_collision(new_node))
58           rrtree.insert_node(nearest_index, new_node);
59           dist=rrtree.checkgoal_offline(new_node);
60       end
61
62
63
64       if(mod(ind, 100) == 0)
65           disp([num2str(rrtree.nodes_added-1) ' nodes in ' num2str(toc)]);
66       end
67
68   end
69
70
71
72   % end timer
73   toc;
74
75   % plot
76   rrtree.findpath();
77
78   plot(XYline.x1,XYline.y1,'black--', 'LineWidth',2);
79   hold on
80   plot(XYline.x2,XYline.y2,'black--', 'LineWidth',2);
81   rrtree.plotpath();
82   hold on
83   p2=rrtree.plotobstacle();
84   figure(2)
85   subplot(5,1,1)
86   plot(rrtree.path(1,:),rrtree.path(2,:));
87   xlabel('x(m)')
88   ylabel('y(m)')
89   subplot(5,1,2)
90   plot(rrtree.path(1,:),rrtree.path(4,:));
91   xlabel('x(m)')
92   ylabel('velocity(m/s)')
93   subplot(5,1,3)
94   plot(rrtree.path(1,:),rrtree.path(5,:));
95   xlabel('x')
96   ylabel('Steering angle(rad)')
97   subplot(5,1,4)
98   plot(rrtree.path(1,:),rrtree.path(6,:));
99   xlabel('x')
100  ylabel('Acceleration(m/s2)')
101  subplot(5,1,5)
102  plot(rrtree.path(1,:),rrtree.path(7,:));
```

```
103 xlabel('x')
104 ylabel('Steering rate (rad/s)')
```

A-1-3 Online RRT Algorithm over Closed-Loop System

```
1  clear all;
2  clc;
3  close all;
4
5  max_iter=20000;
6  max_nodes=max_iter;
7  rand_seed=1;
8
9  dist=0;
10 b=1;
11 t_p=0;
12 a=0;
13 v=66/3.6;
14 start=[150,2.75,0];
15 x_init=[150,2.75,0,60/3.6,0];
16 tree_new=[150,2.75,0,60/3.6,0,0,0,0,0];
17
18
19 conf = struct;
20 conf.delta_goal_point = [50,0.15];          % Radius of goal point
21 conf.delta_near =1.5;                  % Radius for neighboring nodes
22 conf.max_step =1; % Maximum position change when we add a new node to the
       tree
23 conf.W=3; % width/2 of the vehicle
24 conf.L=9; % length/2 of the vehicle
25 conf.U_boundary=[-pi/12;pi/6;-2.17;1.77]; % input constraints
26 path_array=cell(500,2000);
27 path_trajectory=cell(1,1000);
28
29
30
31
32 % plot traffic lanes
33 XYline=struct;
34 XYline.x1=0:0.1:460;
35 XYline.y1=0*XYline.x1+8;
36 XYline.x2=100:0.1:355;
37 XYline.y2=0*XYline.x2+4.5;
38
39
40
41 %%% Starting a timer
42 tic;
43
44 %%%%%%%%%
45
46
47
```

```
48 figure(1)
49 p_old=plot(0,0);
50 hold on
51 plot(XYline.x1,XYline.y1,'black--', 'LineWidth',2);
52 hold on
53 plot(XYline.x2,XYline.y2,'black--', 'LineWidth',2);
54 hold on
55
56
57
58 while (dist==0) % when dist~=0, the goal is found
59    t_p=1+t_p;
60    t=(t_p-1)*0.05;
61    clear rrtree %
62    map=struct('name', 'obstacle23.mat', 'start_point', start, 'goal_point
          ', [230,6.25,0]); % load map
63    variant='RRT_timestop'; % load class
64    rrtree= eval([variant '(rand_seed, max_nodes, map, conf);']); % use
          class
65    rrtree.update_tree(tree_new);
66
67    rrtree.update_ob(a,v,t); % update obstacle at time (t-1);
68    ist=1;
69
70    while(ist==1)
71    for i=1:1:10  % sim the closed-loop 20 times
72     new_node = rrtree.sample_XY(); %3*1 sample configuration space
73     %u=rrtree.sample_U(); %2*1 sample inputs
74
75     [nearest_node,nearest_index] = rrtree.nearest(new_node); %[x,y,theta,
          v] find nearest node in the tree from the sampled node
76
77
78      % 4*1 %initialize closed-loop
79      ref=rrtree.steer(nearest_index,new_node);
80      xref=ref(1);
81      yref=ref(2);
82      simout=sim('V_model_CL','AbsTol','1e-5');
83      new=simout.get('simout');
84      Anode=new.Data';
85      node=Anode(:,end); % pick up the end node[x,y,theta,v,steeringangle,
            acc]
86      new_node=[node;ref];
87
88      if(~rrtree.obstacle_collision(new_node)) %check collision
89        rrtree.insert_node(nearest_index, new_node);% insert the collosion
             -free nodes to the tree
90      end
91
92
93    end
94
95     disp([num2str(rrtree.nodes_added-1) ' nodes in ' num2str(toc)]);
```

```
96
97      path=rrtree.findpath();     %find a shortest path
98      [a,b]=size(path);
99      ist=b;
100
101     end
102
103     path_array{1,t_p}=path;     %store the planned path
104
105     dist=rrtree.checkgoal();    % check wtheter the goal is achieved
106
107
108     start=path(1:3,1);
109     x_init=path(1:5,1)';
110     tree_new=path(:,1)';
111
112     rrtree.plotpath();% plot the path
113     hold on;
114     p2=rrtree.plotobstacle();
115     hold on
116     delete(p_old);
117     p_old=p2;
118
119
120
121 end
122
123
124 % end timer
125 toc;
```

A-1-4 RRT Template

```
1  classdef RRT_timestop < handle
2      properties (SetAccess = private)
3          W                   %width/2
4          L                   % Length/2
5          tree                % Array stores position information of states
6          parent              % Array stores relations of nodes
7          children            % Number of children of each node
8          free_nodes          % Indices of free nodes
9          free_nodes_ind      % Last element in free_nodes
10         cost                % Cost between 2 connected states
11         cumcost             % Cost from the root of the tree to the given
               node
12         XY_BOUNDARY         % [min_x max_x min_y max_y]
13         goal_point          % Goal position
14         start_point         % Start position
15         delta_goal_point    % Radius of goal position region
16         delta_near          % Radius of near neighbor nodes
17         nodes_added         % Keeps count of added nodes
18         max_step            % The length of the maximum step while adding
               the node
```

```
19          obstacle              % Obstacle information
20          velocity_ob          % Dynamic Obstacles Information
21          best_path_node        % The index of last node of the best path
22          goal_reached
23          max_nodes
24          U_boundary
25          obstacle_cur
26
27
28
29
30          %%% temporary variables
31          compare_table
32          index
33          list
34          %%%
35          path
36
37          backtrace_index
38
39
40
41      end
42
43      methods
44          %%%%%%%
45
46          function  main= RRT_timestop(rand_seed, max_nodes, map, conf) %
                class constructor
47              main.W = conf.W;
48              main.L = conf.L;
49              max_nodes = int32(max_nodes);
50              main.max_nodes = max_nodes;
51              rng(rand_seed);
52              main.tree = zeros(9, max_nodes);
53              main.parent = zeros(1, max_nodes);
54              main.children = zeros(1, max_nodes);
55              main.free_nodes = zeros(1, max_nodes);
56              main.free_nodes_ind = 1;
57              main.cost = zeros(1, max_nodes);
58              main.cumcost = zeros(1,max_nodes);
59              main.XY_BOUNDARY = zeros(4,1);
60              main.tree(1:3,1) = map.start_point;
61              main.tree(4,1) = 60/3.6;
62              main.goal_point = map.goal_point;
63              main.start_point=map.start_point;
64              main.delta_goal_point = conf.delta_goal_point;
65              main.delta_near = conf.delta_near;
66              main.nodes_added = uint32(1);
67              main.max_step = conf.max_step;
68              main.best_path_node = -1;
69              main.goal_reached = false;
70              main.load_map(map.name);
```

```
71              main.U_boundary=conf.U_boundary;
72              %%% temp var-s initialization
73              main.compare_table = zeros(1, max_nodes);
74              main.index = zeros(1, max_nodes);
75              main.list = 1:max_nodes;
76              %%%%
77              main.path=zeros(9,max_nodes/10);
78
79          end
80
81          %%%%%%%
82
83          function position= sample_XY(main) % sample x,y theta
84
85              %position = [main.XY_BOUNDARY(2) - main.XY_BOUNDARY(1); main.
                    XY_BOUNDARY(4) - main.XY_BOUNDARY(3); pi/3] .* rand(3,1)
                    ...
86                  %+ [main.XY_BOUNDARY(1);main.XY_BOUNDARY(3);-pi/6];
87              x0=main.goal_point(1);
88              y0=main.goal_point(2);
89              theta0=main.goal_point(3);
90              sigma_r=3;
91              sigma_t=pi/8;
92              nr=random('Normal',0,sigma_r,1,1);
93              nt=random('Normal',0,sigma_t,1,1);
94              r=sigma_r*abs(nr)+1;
95              theta=sigma_t*abs(nt)+theta0;
96              position=[x0-r*cos(theta);y0-r*sin(theta);theta];%
97           end
98
99
100         %%%%%%%%%%%%
101
102         function input=sample_U(main) % sample steering angle, a
103             u1=(main.U_boundary(2)-main.U_boundary(1))*rand(1,1)+main.
                    U_boundary(1);
104             u2=(main.U_boundary(4)-main.U_boundary(3))*rand(1,1)+main.
                    U_boundary(3);
105             input=[u1;u2];
106         end
107
108         %%%%%%%%
109
110         function [node_nearst,node_index] = nearest(main, new_node) %
                find nearest node
111             % find the nearest node to the given node, euclidian distance
                     is used
112             main.compare_table(1:(main.nodes_added)) = sum((main.tree
                    (1:2, 1:(main.nodes_added)) - repmat(new_node(1:2),1,main.
                    nodes_added)).^2) + sum((main.tree(3:main.nodes_added) -
                    new_node(3)).^2);
```

```
113             [main.compare_table(1:(main.nodes_added)), main.index(1:(main
                    .nodes_added))] = sort(main.compare_table(1:(main.
                    nodes_added)));
114             node_index = main.index(1);
115             node_nearst=main.tree(1:5,node_index);
116 
117             return;
118         end
119 
120         %%%%%%%%
121 
122          function position = steer(main, nearest_index, new_node_position
                )
123 
124             if(norm(new_node_position(1:2) - main.tree(1:2, nearest_index
                    )) > main.max_step)
125                 theta = atan((new_node_position(2) - main.tree(2,
                        nearest_index))/(new_node_position(1) - main.tree(1,
                        nearest_index)));
126                 position = main.tree(1:2, nearest_index) ...
127                     + [sign((new_node_position(1) - main.tree(1,
                            nearest_index))) * main.max_step * cos(theta); ...
128                     sign((new_node_position(2) - main.tree(2,
                            nearest_index))) * main.max_step * abs(sin(theta))
                            ];
129             else
130                 position = new_node_position(1:2);
131             end
132 
133 
134         end
135 
136         %%%%%%%%
137 
138         function position = steer_off(main, nearest_index,
                new_node_position)
139 
140             if(norm(new_node_position(1:2) - main.tree(1:2, nearest_index
                    )) > main.max_step)
141                 theta = new_node_position(3);
142                 position = main.tree(1:2, nearest_index) ...
143                     + [sign((new_node_position(1) - main.tree(1,
                            nearest_index))) * main.max_step * cos(theta); ...
144                     sign((new_node_position(2) - main.tree(2,
                            nearest_index))) * main.max_step * abs(sin(theta))
                            ];
145                 position=[position;new_node_position(3:7)];
146             else
147                 position = new_node_position;
148             end
149 
150 
151         end
```

```
152
153
154
155
156
157
158         %%%%%%
159         function load_map(main, map_name) % load map
160             % function loads '.mat' file with obstacle information and
                    the
161             % size of the map
162             map_path = 'maps/';
163             main.obstacle = load([map_path map_name], 'num', 'output', '
                    x_constraints', 'y_constraints','num_s','num_d','
                    dynamic_ob','static_ob');
164             main.obstacle.vert_num = zeros(main.obstacle.num,1);
165             main.obstacle.m = cell(main.obstacle.num,1);
166             main.obstacle.b = cell(main.obstacle.num,1);
167             main.obstacle.r = zeros(main.obstacle.num,1);
168             main.obstacle.r_sqr = zeros(main.obstacle.num,1);
169             main.obstacle.cir_center = cell(main.obstacle.num,1);
170             main.XY_BOUNDARY = [main.obstacle.x_constraints main.obstacle
                    .y_constraints];
171
172         end
173
174         %%%%%%%%
175
176         function collision = obstacle_collision(main,new_node) % check
                collision
177
178             collision = false;
179             theta = new_node(3);
180
181             if (mod(theta, pi/2) == 0)
182                 theta = theta - 1;
183             end
184
185
186             % omit any operations if there is no obstacles
187             if main.obstacle.num == 0
188                 return;
189             end
190
191             for obs_ind = 1:main.obstacle.num
192
193
194              vertex1 = [new_node(1)-cos(theta)*main.L/2-sin(theta)*main.W
                        /2,new_node(2)-sin(theta)*main.L/2+cos(theta)*main.W/2];
195              vertex2 = [new_node(1)+cos(theta)*main.L/2-sin(theta)*main.W
                        /2,new_node(2)+sin(theta)*main.L/2+cos(theta)*main.W/2];
196              vertex3 = [new_node(1)+cos(theta)*main.L/2+sin(theta)*main.W
                        /2,new_node(2)+sin(theta)*main.L/2-cos(theta)*main.W/2];
```

```
197             vertex4 = [new_node(1)-cos(theta)*main.L/2+sin(theta)*main.W
                    /2,new_node(2)-sin(theta)*main.L/2-cos(theta)*main.W/2];
198
199
200                 if isintersect_3dof(main.obstacle_cur{obs_ind}, [vertex1;
                        vertex2]) == 1 ...
201                         || isintersect_3dof(main.obstacle_cur{obs_ind}, [
                            vertex2; vertex3]) == 1 ...
202                         || isintersect_3dof(main.obstacle_cur{obs_ind}, [
                            vertex3; vertex4]) == 1 ...
203                         || isintersect_3dof(main.obstacle_cur{obs_ind}, [
                            vertex4; vertex1]) == 1
204                     collision = true;
205                     return;
206                 end
207             end
208
209         end
210
211
212
213         %%%%%%%
214
215         function new_node_ind = insert_node(main, parent_node_ind,
                new_node) %insert collision-free node
216             % method insert new node in the tree
217             main.nodes_added = main.nodes_added + 1;
218             main.tree(:, main.nodes_added) = new_node;          % adding
                    new node position to the tree
219
220             main.parent(main.nodes_added) = parent_node_ind;            %
                     adding information about parent-children information
221             main.children(parent_node_ind) = main.children(
                    parent_node_ind) + 1;
222             main.cumcost(main.nodes_added) = main.cumcost(parent_node_ind
                    ) + main.cost(main.nodes_added);    % cummulative cost
223             new_node_ind = main.nodes_added;
224         end
225
226         %%%%%%%
227         function dist=checkgoal(main)
228
229             xv=[main.goal_point(1)-main.delta_goal_point(1);main.
                    goal_point(1)-main.delta_goal_point(1);main.goal_point(1)+
                    main.delta_goal_point(1);main.goal_point(1)+main.
                    delta_goal_point(1);main.goal_point(1)-main.
                    delta_goal_point(1)];
230             yv=[main.goal_point(2)-main.delta_goal_point(2);main.
                    goal_point(2)+main.delta_goal_point(2);main.goal_point(2)+
                    main.delta_goal_point(2);main.goal_point(2)-main.
                    delta_goal_point(2);main.goal_point(2)-main.
                    delta_goal_point(2)];
231             xq=main.path(1,:)';
```

```
232             yq=main.path(2,:)';
233             [in,on]=inpolygon(xq,yq,xv,yv);
234
235             dist=numel(xq(in))+numel(xq(on));
236
237         end %stop the algorithm
238
239         %%%%%%%
240
241         function dist=checkgoal_offline(main,new_node)
242
243             xv=[main.goal_point(1)-main.delta_goal_point(1);main.
                    goal_point(1)-main.delta_goal_point(1);main.goal_point(1)+
                    main.delta_goal_point(1);main.goal_point(1)+main.
                    delta_goal_point(1);main.goal_point(1)-main.
                    delta_goal_point(1)];
244             yv=[main.goal_point(2)-main.delta_goal_point(2);main.
                    goal_point(2)+main.delta_goal_point(2);main.goal_point(2)+
                    main.delta_goal_point(2);main.goal_point(2)-main.
                    delta_goal_point(2);main.goal_point(2)-main.
                    delta_goal_point(2)];
245             xq=new_node(1);
246             yq=new_node(2);
247             theta=new_node(3);
248
249              [in,on]=inpolygon(xq,yq,xv,yv);
250              dist1=numel(xq(in))+numel(xq(on));
251              if -0.1<=theta<=0.1 && dist1~=0
252                 dist=1;
253              else
254                   dist=0;
255               end
256
257
258          end %stop the algorithm
259
260
261
262
263         %%%%%%%%%%
264
265         function path=findpath(main)  %%Find the optimal path to the goal
                and plot
266             % finding all the point which are in the desired region
267
268             distances = zeros(main.nodes_added, 2);
269             distances(:, 1) = sum((main.tree(1:3,1:(main.nodes_added)) -
                    repmat(main.goal_point(1:3)', 1, main.nodes_added)).^2);
270             distances(:, 2) = 1:main.nodes_added;
271             distances = sortrows(distances, 1);
272
273             nearest_node_index = distances(1,2);
274
```

```
275             %backtracing the path
276             current_index = nearest_node_index;
277             path_iter = 1;
278             backtrace_path = zeros(1,1);
279             while(current_index ~= 1)
280                 backtrace_path(path_iter) = current_index;
281                 path_iter = path_iter + 1;
282                 current_index = main.parent(current_index);
283             end
284             backtrace_path(path_iter) = current_index;
285
286             main.path=main.tree(:,backtrace_path);
287
288             path=main.path;
289
290
291         end
292
293
294         %%%%%%%%%%
295         function update_ob(main,a,v, t) %update the obstacle
296
297             cur_obstacle=cell(1,main.obstacle.num);
298
299             for j=1:main.obstacle.num_s
300             cur_obstacle{j}=main.obstacle.static_ob{j};
301             end
302
303             obstacle_dy=cell(1,main.obstacle.num_d);
304            for i=1:main.obstacle.num_d
305                obstacle_dy{i}(:,1)=main.obstacle.dynamic_ob{i}(:,1)+v*t
                       +0.5*a*t^2;
306                if min(obstacle_dy{i}(:,1))>=main.XY_BOUNDARY(2)
307                 n=floor(max(obstacle_dy{i}(:,1))/main.XY_BOUNDARY(2));
308                 obstacle_dy{i}(:,1)=obstacle_dy{i}(:,1)-n*main.
                        XY_BOUNDARY(2);
309                else
310                 obstacle_dy{i}(:,1)=obstacle_dy{i}(:,1);
311                end
312
313                obstacle_dy{i}(:,2)=main.obstacle.dynamic_ob{i}(:,2);
314                cur_obstacle{main.obstacle.num_s+i}=obstacle_dy{i};
315            end
316
317
318             main.obstacle_cur=cur_obstacle;
319
320
321         end
322
323         %%%%%%%%
324         function update_tree(main,new)
325             main.tree(:,1)=new;
```

```
326         end
327
328
329
330         %%%%%%%%%
331
332
333         function plotpath(main) % plot path
334             [m,n]=size(main.path);
335             for i=1:n
336             vertex1 = [main.path(1,i)-cos(main.path(3,i))*main.L/2-sin(
                    main.path(3,i))*main.W/2,main.path(2,i)-sin(main.path(3,i)
                    )*main.L/2+cos(main.path(3,i))*main.W/2];
337             vertex2 = [main.path(1,i)+cos(main.path(3,i))*main.L/2-sin(
                    main.path(3,i))*main.W/2,main.path(2,i)+sin(main.path(3,i)
                    )*main.L/2+cos(main.path(3,i))*main.W/2];
338             vertex3 = [main.path(1,i)+cos(main.path(3,i))*main.L/2+sin(
                    main.path(3,i))*main.W/2,main.path(2,i)+sin(main.path(3,i)
                    )*main.L/2-cos(main.path(3,i))*main.W/2];
339             vertex4 = [main.path(1,i)-cos(main.path(3,i))*main.L/2+sin(
                    main.path(3,i))*main.W/2,main.path(2,i)-sin(main.path(3,i)
                    )*main.L/2-cos(main.path(3,i))*main.W/2];
340              plot([vertex1(1),vertex2(1)],[vertex1(2),vertex2(2)],'y')
341              hold on
342              plot([vertex2(1),vertex3(1)],[vertex2(2),vertex3(2)],'y')
343              hold on
344              plot([vertex3(1),vertex4(1)],[vertex3(2),vertex4(2)],'y')
345              hold on
346              plot([vertex4(1),vertex1(1)],[vertex4(2),vertex1(2)],'y')
347              hold on
348             end
349
350
351             plot(main.path(1,:),main.path(2,:),'*b','LineWidth', 1);
352             set(gcf(), 'Renderer', 'opengl');
353         end
354
355         %%%%%%%%%
356         function p2=plotobstacle(main)
357             p2=zeros(12,1);
358             for k = 1:main.obstacle.num_s
359                 p1 = fill(main.obstacle_cur{k}(1:end, 1), main.
                        obstacle_cur{k}(1:end, 2), 'black');
360                 set(p1,'HandleVisibility','off','EdgeAlpha',0);
361             end
362
363             for i=main.obstacle.num_s+1:main.obstacle.num
364                 p2(i)=fill(main.obstacle_cur{i}(1:end, 1), main.
                        obstacle_cur{i}(1:end, 2), 'black');
365             end
366 %
367
368         end
```

```
369
370
371         %%%%%%%
372
373
374
375     end
376
377
378
379     methods(Static)
380         function plot_circle(x, y, r)
381             t = 0:0.001:2*pi;
382             cir_x = r*cos(t) + x;
383             cir_y = r*sin(t) + y;
384             plot(cir_x, cir_y, 'r-', 'LineWidth', 1.5);
385         end
386     end
387 end
```

A-1-5 Runga-Kutta Algorithm

```
1  function position=rungakutta(x_init,u)
2
3  h=0.02; %step size
4  t(1) = 0;
5  w(:,1) = x_init;      %initial conditions [x;y;theata;v;delta]
6
7
8  for i = 1:5
9
10     k1 = h*f(t(i), w(:,i),u);
11     k2 = h*f(t(i)+h/2, w(:,i)+0.5*k1,u);
12     k3 = h*f(t(i)+h/2, w(:,i)+0.5*k2,u);
13     k4 = h*f(t(i)+h, w(:,i)+k3,u);
14     w(:,i+1) = w(:,i) + (k1 + 2*k2 + 2*k3 + k4)/6;
15     t(i+1) = t(1) + i*h;
16
17 end
18 position=[w(:,4);u];%[x,y,theta,v,steering angle,acceleration]
19 end
```

A-1-6 Vehicle Model

```
1  function dy = f(t,y,u) % y=[x,y,theta,v,delta]
2  l=5;
3  dy = [y(4)*cos(y(3)); %x
4        y(4)*sin(y(3));  %y
5        2*y(4)/l*tan(y(5)/(1+(y(4)/216)));%thtea
6        y(6); %v
7        (u(2,1)-y(5))/1.5;%delta
8        (u(1,1)-y(6))/1.2];%a
9  end
```

A-2 SIMULINK Model

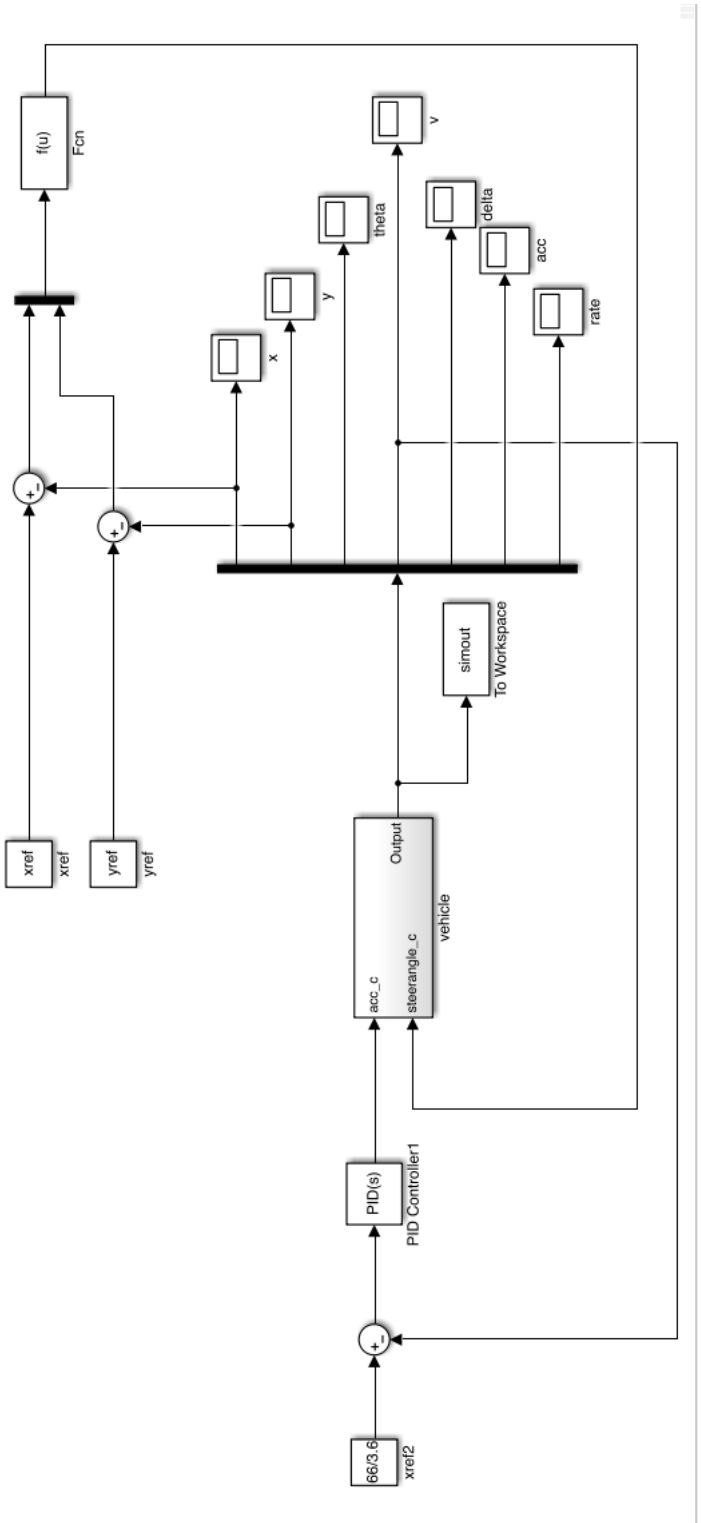


Figure A-1: SIMULINK Model.

Bibliography

[1] "Gross domestic (gdp) attribute to for-hire transportation services," *Bureau of Transportation Statistics, United States Department of Transportation*, 2014.

[2] J. R. B. N. Janson, W. Awad and B. P. J. Kononov, "Truck accident at freeway ramps: Data analysis and high-risk site identification," *Department of Civil Engineering, University of Colorado at Denver, Colorado Department of Transportation*, 1998.

[3] B. L. Bowman and H. S. Lum, "Examination of truck accident on urban freeway," *Ite Journal*, 1990.

[4] M. Luijten, *Lateral dynamic behaviour of articulated commercial vehicles.* PhD thesis, Master Thesis 2010, Eindhoven University of Technology, 2010.

[5] S. Gottschalk, *Collision queries using oriented bounding boxes.* PhD thesis, The University of North Carolina at Chapel Hill, 2000.

[6] B. W. Bequette, *Process control: modeling, design, and simulation.* Prentice Hall Professional, 2003.

[7] J. Zhong, "Pid controller tuning: a short tutorial," 2006.

[8] N. van Duijkeren, *Real-time Path Planning and Obstacle Avoidance for a Long Heavy Vehicle Combination in a Lane-change Maneuver.* PhD thesis, 2013.

[9] "Transport safety facts 2012 data," *Department of Transportation, National Highway Traffic Safety Administration, U.S.*, 2012.

[10] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[11] L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," *IEEE Transactions on, Robotics and Automation*, vol. 14, no. 1, pp. 166–171, 1998.

[12] S. M. LaValle and J. J. Kuffner Jr, "Rapidly-exploring random trees: Progress and prospects," 2000.

[13] T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Transactions on, Computers*, vol. 100, no. 2, pp. 108–120, 1983.

[14] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. P. How, and G. Fiore, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on, Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[15] T. D. Gillespie, "Fundamentals of vehicle dynamics," tech. rep., SAE Technical Paper, 1992.

[16] D. Assanis, Z. Filipi, S. Gravante, D. Grohnke, X. Gui, L. Louca, G. Rideout, J. Stein, and Y. Wang, "Validation and use of simulink integrated, high fidelity, engine-in-vehicle simulation of the international class vi truck," 2000.

[17] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, 1993.

[18] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[19] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *The International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, 1991.

[20] Z. Shiller, "Off-line and on-line trajectory planning," in *Motion and Operation Planning of Robotic Systems*, pp. 29–62, Springer, 2015.

[21] R. Pepy, A. Lambert, and H. Mounier, "Path planning using a dynamic vehicle model," in *Information and Communication Technologies, 2006. ICTTA'06. 2nd*, vol. 1, pp. 781–786, IEEE.

[22] J. W. Boyse, "Interference detection among solids and surfaces," *Communications of the ACM*, vol. 22, no. 1, pp. 3–9, 1979.

[23] G. T. Toussaint, "Solving geometric problems with the rotating calipers," in *Proc. IEEE Melecon*, vol. 83, p. A10, 1983.

[24] P. M. Hubbard, "Approximating polyhedra with spheres for time-critical collision detection," *ACM Transactions on Graphics (TOG)*, vol. 15, no. 3, pp. 179–210, 1996.

[25] L. Zhang, "The research and realization of collision detection in virtual reality," *Journal of Communication and Computer*, vol. 8, no. 8, pp. 693–696, 2011.

[26] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 171–180, ACM, 1996.

[27] J. T. Klosowski, "k-dops as tighter bounding volumes for better occlusion performance (skapps_0030),"

[28] S. A. Ehmann and M. C. Lin, "Accurate and fast proximity queries between polyhedra using convex surface decomposition," in *Computer Graphics Forum*, vol. 20, pp. 500–511, Wiley Online Library, 2001.

[29] Q. J, *Control Methods Study of the Autonomous Vehicle*. PhD thesis, Beijing University of Technology, 2009.

[30] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. P. How, "Motion planning in complex environments using closed-loop prediction," 2008.

[31] K. J. Astrom, *Control System Design*. 2002.

[32] G. M. Nagrath J, *Control System Engineering*. New Age International Publications, 2002.

[33] Y. Shan, W. Yang, C. Chen, J. Zhou, L. Zheng, and B. Li, "Cf-pursuit: A pursuit method with a clothoid fitting and a fuzzy controller for autonomous vehicles," *International Journal of Advanced Robotic Systems*, vol. 12, 2015.

[34] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," tech. rep., DTIC Document, 1992.

[35] W. S. Levine, *Control System Fundamentals*. CRC press, 1999.

[36] A. Mehar, S. Chandra, and S. Velmurugan, "Speed and acceleration characteristics of different types of vehicles on multi-lane highways,"

[37] C. Kou and R. Machemehl, "Modeling driver behavior during merge maneuvers," tech. rep., 1997.

Glossary

List of Acronyms

BVs: Bounding Volumes

DARPA: Defense Advanced Research Projects Agency

RRT: Rapidly Exploring Random Tree

PRM: Probabilistic Road Maps

AABB: Axis-Aligned Bounding Box

OBB: Oriented Bounding Box

COG: Coordinates of Centre of Gravity

PID: Proportional-Integral-Derivative

PI: Proportional-Integral

PF: Putative Follower

PL: Putative Leader

RK: Runge-Kutta

List of Symbols

X: Coordinate in x-axis in inertial frame

Y: Coordinate in y-axis in inertial frame

θ: Orientation angle in inertial frame

V: Velocity

δ: Steering angle

L: Length of wheelbase

V_{char}: Characteristic velocity

g: Gravitational acceleration

$C_{\alpha f}$: Cornering stiffness of the front wheel

$C_{\alpha r}$: Cornering stiffness of the rear wheel

F_z: Load on the tires

K: Understeer gradient

a: Acceleration

r: Steering rate

L_f: Distance from front axle to cog laden

L_r: Distance from rear axle to cog laden

L_d: Look-ahead distance

Δt: Updating time

ΔT: Integration time