# State Lattice-based Motion Planning for Autonomous On-Road Driving

**Dissertation**

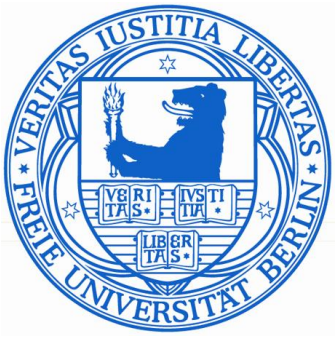
zur Erlangung des akademischen Grades

der Doktorin der Naturwissenschaften (Dr. rer. nat)

am Institut für Informatik

des Fachbereichs Mathematik und Informatik

der Freien Universität Berlin




vorgelegt von

**Shuiying Wang**

Berlin, 2015

Gutachter:

Prof. Dr. Raúl Rojas
Institut für Informatik
Freie Universität Berlin

Prof. Dr. Manfred Hild
Institut für Elektrotechnik, Mechatronik, Optometrie
Beuth Hochschule für Technik Berlin

# *Preface*

Since DARPA Urban Challenge 2007 (DUC), the development of autonomous vehicles has attracted increasing attention from both academic institutes and the automotive industry. It is believed that autonomous vehicles sophisticated and reliable enough would redefine mobility. The motion planner and sensor simulation presented in this thesis are intended to contribute to this prospect.

The task of a motion planner for autonomous on-road vehicles is to generate a trajectory of motions for the vehicle to follow. The proposed motion planner employs a state lattice to construct a large variety of candidate trajectories and selects the best constraint-abiding one based on a set of cost criteria. The parallel computer architecture of CUDA is exploited to construct and evaluate the trajectories efficiently. The spatial planning horizon of the proposed planner can contain multiple segments with different widths. This feature helps the planner to adapt to various road layouts easily and to generate more consistent plans.

During the construction of the state lattice, acceleration profiles are associated with the path segments to generate trajectory segments. Acceleration cubic polynomials and constant accelerations are applied in the proposed planner. The adopted association scheme makes it possible to span one acceleration profile over several trajectory segments, which, together with the applied smooth acceleration profiles, helps to enhance the feasibility of the trajectories. In the construction of cost maps of obstacles for the evaluation of the trajectories, the obstacles need to be dilated to compensate for the vehicle shape. A novel approach is proposed to analyse the sufficiency of the dilation strategy implemented in this work from the perspective of excluding all the trajectories that are practically not traversable.

The scanning sensors are also simulated to increase the realistic level of the simulation experiments. Programmable shaders in the rendering pipeline of OpenGL are manipulated to record sensor-related data. Such implementation takes advantage of the parallel computer architecture of the GPU and thus enhances the computational efficiency of the generation process of the simulated sensor data. A novel macro-micro approach is proposed which can increase the accuracy of the simulated sensor data. Finally, the proposed planner is evaluated in a variety of simulated traffic scenarios. Given proper guidances from a behaviour planning layer, the proposed planner can generate reasonable plans in most scenarios.

# *Vorwort*

Seit der DARPA Urban Challenge 2007 (DUC) bekommt die Entwicklung autonomer Fahrzeuge erhöhte Aufmerksamkeit aus der Forschung und aus der Automobilindustrie. Es wird vermutet, dass ausgereifte und zuverlässige autonome Fahrzeuge unsere Mobilität neu definieren würden. Der in dieser Arbeit vorgestellte Bewegungsplaner und die Sensorsimulation beabsichtigten zu dieser Entwicklung beizutragen.

Die Aufgabe eines für autonomes Fahren geeigneten Bewegungsplaners ist es, eine Trajektorie von Bewegungen zu generieren, sodass das Auto auf diese Trajektorie folgen kann. Der vorgeschlagene Bewegungsplaner wendet ein Zustandsgitter an, um viele unterschiedliche Kandidaten-Trajektorien zu konstruieren, und wählt die den Beschränkungen entsprechende beste auf der Grundlage einer Reihe von Kostenkriterien aus. Die parallele Rechnerarchitektur von CUDA wird für die Erzeugung und die Auswertung der Trajektorien effizient eingesetzt. Der räumliche Planungshorizont dieses Planers kann aus mehreren Segmenten mit unterschiedlichen Breiten bestehen. Diese Eigenschaft hilft dem Planer, sich auf verschiedenen Fahrbahntypen anzupassen und konsistentere Pläne zu generieren.

Während der Berechnung der Zustandsmatrix werden die Ränder der Fahrbahn den Beschleunigungsprofilen zugeordnet, um die Kanten der Trajektorien zu erzeugen. Kubisch-polynomielle und konstante Beschleunigung werden in dem vorgeschlagenen Planer angewendet. Das angewendete Assoziationsschema macht es möglich, ein Beschleunigungsprofil über mehrere Kanten einer Trajektorie auszudehnen. Das hilft, zusammen mit den angewendeten glatten Beschleunigungsprofilen, die Ausführbarkeit der Trajektorien zu verbessern. Bei der Erstellung von Kostenkarten von Hindernissen für die Auswertung der Trajektorien müssen die Hindernisse erweitert werden, um den Umriss des Fahrzeugs zu kompensieren. Als neuer Ansatz wird vorgeschlagen, die Angemessenheit der in dieser Arbeit implementierten Ausweitungsstrategie aus der Perspektive des Ausschlusses aller praktisch nicht befahrbahren Trajektorien zu analysieren.

Die LiDAR und Radarsensoren werden ebenfalls simuliert, um den Realismus der Simulationsexperimente zu erhöhen. Programmierbare Shader in der Rendering- Pipeline von OpenGL werden verändert, um Sensordaten aufzuzeichnen. Diese Umsetzung nutzt die Vorteile der parallelen Rechnerarchitektur der GPU und verbessert so die Recheneffizienz bei der Generierung der simulierten Sensordaten. Ein neuer Makro-Mikro-Ansatz wird vorgeschlagen, um die Genauigkeit der simulierten Sensordaten zu erhöhen. Schließlich wird der vorgeschlagene Planer in einer Vielzahl von simulierten Verkehrssituationen ausgewertet. Mit gegebener Orientierung von einer Verhaltensplanungsschicht kann der vorgeschlagene Planer in den meisten Szenarien vernünftige Pläne generieren.

# *Acknowledgements*

Many thanks go to my advisor, Prof. Dr. Raúl Rojas, for trusting me to take on the work of motion planning and sensor simulation. He encouraged me to investigate into the simplest forms of the problems, which turns out to be very beneficial in both research and thesis writing. Sincere thanks also go to Prof. Dr. Manfred Hild for reviewing the thesis and offering instructive suggestions for further improvement.

I thank Till Zoppke, Miao Wang and Fabian Wiesel for helping me in getting familiar with the software system of the autonomous vehicle project during the first year of my doctoral study.

Thanks go to Till Zoppke again, and Manuel Schwabe, Sebastian Hempel, Michael Schnürmacher, Ernesto Tapia, Tobias Langner, Bennet Fischer, for patiently explaining to me the functions of the radar sensor, the Velodyne LiDAR sensor, the IBEO LUX LiDAR sensor and the video cameras. The sensor simulation, which is one part of this work, cannot be completed without their help. Thanks to Steffen Heinrich for collaborating with me in improving the simulator. Thanks to Marco Block-Berlitz for introducing me to the topic of sensor simulation.

I thank Miao Wang again, and Daniel Göhring, Tinosch Ganjineh and Fritz Ulbrich for their explanations about the planning system of MIG and their advices on the improvements of the planner. Fritz Ulbrich also helps me a lot in solving many programming problems, which I appreciate sincerely. Thanks to Paul Czerwionka for helping me in understanding the RNDF road model. Thanks go to Georg Bremer for his help and patience in solving many weird programming bugs that I was confronted to during the implementation of the motion planner.

I also thank all the contributors to the project of autonomous car MIG. Without the MIG system built through their incredibly excellent and hard work, the work presented in this thesis has no chance to exist.

Many thanks go to Cedric De Brito, Omar Mendoza Montoya, Till Zoppke and Miao Guo for helping me in correcting the grammatical and logical errors of the manuscript of this thesis.

I owe a debt of gratitude to my parents who are always there supporting and caring for me. Thanks to my other family members and my friends for encouraging me to continue to make progress. I thank China Scholarship Council for supporting me financially.

Thanks finally to my committee for spending their time and efforts on assessing my work.

# Contents

# Chapter 1

# Introduction

Since DARPA Urban Challenge 2007 (DUC), the development of autonomous vehicles has attracted increasing attention from both academic institutes and the automotive industry. It is believed that autonomous vehicles sophisticated and reliable enough would redefine mobility. The motion planner and sensor simulation presented in this thesis are intended to contribute to this prospect.

In on-road traffic scenarios, a motion planner capable of generating flexible and feasible trajectories within short planning cycle is necessary for autonomous vehicles to perform sophisticated and reliable driving behaviours. Motion planning described in this thesis conducts its search for the best constraint-abiding trajectory in road-adapted state lattice. Such method belongs to the domain of search-based planning, which requires constructing a graph and searching for the optimal trajectory along the constructed graph. As the trajectory generated in this way is piecewise, discontinuity in jerk, acceleration and velocity at the switching points challenges trajectory smoothness. Trajectory smoothness is essential for passenger comfort, energy efficiency and ease of tracking. The acceleration profiles (i.e., acceleration cubic polynomials and constant accelerations) applied in the proposed planner and the corresponding method of associating the acceleration profiles with the path segments can guarantee a high level of trajectory smoothness.

Experiments are crucial for the development of high quality motion planners and thus of effective autonomous driving systems. The experiments involving motion planning reported in this thesis are carried out in a simulation environment. The simulation of scanning sensors (laser scanners and radars) is introduced into the experiments. In this way, the flexibility and reliability of the planner can be tested and verified under more realistic perception limitations. Another part of this thesis presents algorithms for simulated sensor data generation based on shader programming.

## 1.1 Motivation

Autonomous cars can save our efforts and fatigue of driving, increase road safety, spare energy and enhance productivity. To do this, they need sensors to understand the surrounding traffic scenario. Based on the traffic information and their next goal, they reason about their movements for the next several seconds. The cars will then track the planned movements under the help of a controller. Motion planning is all about solving for such a trajectory of movements for the cars to follow, which is an essential component of an autonomous car system.

Simulation for testing autonomous on-road vehicles encompasses vehicle physics simulation, traffic environment modelling and sensor simulation. Simulation in general and sensor simulation in particular is very important in testing and evaluating the performance of autonomous driving systems.

In this section, motion planning and simulation together with their implementation context, i.e., autonomous vehicles, are motivated as problem domains of significant meaning.

### 1.1.1 Societal Impact of Autonomous Vehicles

Mobility is an essential element in the modern society. Autonomous vehicles are all about mobility optimization. Ideally, such vehicles should be able to manoeuvre in both structured and unstructured environments. Their driving operations are completely autonomous, i.e., they are calculated and conducted without any intervention from a human driver. Their travel plans and behaviours are optimal and satisfy necessary constraints. The criteria of optimality may include but are not limited to, minimum risk, time and energy efficiency, and comfort. The constraints are imposed by the requirements of collision avoidance, traffic rules and limitations of the vehicle's physics, etc.

Figure 1.1 shows an example of an autonomous car, MIG(Made In Germany). The sensors simulated in this work including laser scanners and radars are highlighted on it. The algorithms of motion planning and simulation presented in this thesis are implemented in the software system of MIG. More details about MIG can be found in [1].

Commuting is an important aspect of modern societal life that demands a lot of our precious time. In Flanders, Belgium, e.g., 80% of commuting trips occur by cars [2]. Besides the waste of time, the anxiety, stress and tiredness that people might suffer during the commuting trips would impact their state of health to some extent [3]. Such kind of mobility goes completely against efficiency and well being of human beings which are crucial for modern society. By autonomous driving without the need of a human driver, autonomous vehicles would allow commuters to just lean back and enjoy doing everything that is possible in a car.

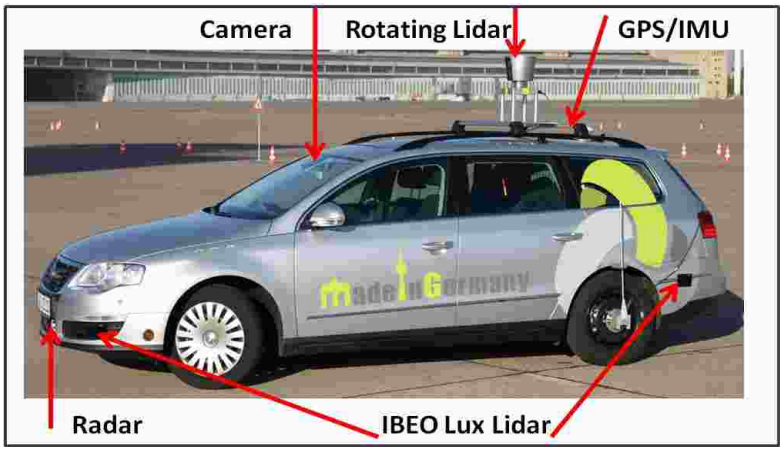



Figure 1.1: The sensor system on an autonomous car.

Traffic congestion is a big problem that challenges the modern society. It is reported that commuters of the USA spend 38 hours a year stuck in traffic with the situations in bigger cities even worse [4]. The congestion "invoice" of extra time and fuel in 498 urban areas amounted to $121 billion in 2011 [4]. It is suggested in [5] that the fact that the desire of human driver for smooth and comfortable driving goes beyond the avoidance of accidents is in part responsible for the occurrence of congested traffic. Autonomous vehicles can help to avoid traffic jam by replacing the human driver with smarter robot capable of generating more flexible, safe and efficient trajectories.

Road safety is a major societal issue. According to statistics provided by mobility and transport administration of European Commission, in 2011, more than 30,000 people died on the roads of the European Union. Even in non-fatal accidents, besides the expenses of loss caused by the traffic accidents, the trauma induced therein can be a long lasting suffer for both the related individuals and the society as a whole. According to [6], human error is thought to be the main factor for 75% of all ground vehicle crashes. Active safety systems that can be found in modern cars such as brake assistant, intelligent speed adaptation, forward collision avoidance and lane departure warning can help to avoid accidents to some extent. However, as mentioned in [6], the active safety system and the human driver are in fact competing against each other, which implicates that the active safety system might fail when the driver is "too crazy". To be more explicit, situations might occur where any reactions of the car cannot avoid the collision. The vehicle states in such scenarios are defined as Inevitable Collision States (ICS) in [7]. Therefore, a complete "cautious" autonomous driving without any intervention from error-prone human drivers is necessary. In this way, many critical situations can be avoided "actively" and thus the safety of on-road traffic can be largely improved. It is noteworthy that sufficient traffic information is a must for safe autonomous driving. In that sense, the recently promoted concepts of Vehicle to Vehicle (V2V) and Vehicle

to Infrastructure (V2I) communications would be of great help in complementing the limited perception of current autonomous vehicles.

Autonomous vehicles also contribute to a mobility of environmental care and energy efficiency. The report on energy consumption and carbon dioxide ($CO_2$) emissions of road transport from Federal statistical office of Germany [8] points out that in 2008 the $CO_2$ emissions of road transport in Germany was 17.4% of the entire $CO_2$ emissions( LULUF (Land Use, Land-Use change and Forestry) excluded). $CO_2$ accounts as one of the main contributors of the notorious green house effect. It also mentions that the road transport has a share of 24.3% of the final energy consumption as a whole in 2007, which concerns the sustainability of energy. Autonomous vehicles that are able to generate efficient travel plans with less pollution are of great meaning to the harmony between the human society and the nature.

Perhaps the ultimate goal of human society in terms of freedom is to conquer physical limitations of human beings caused by ageing and diseases. While the medical solution can still not be available in the near future, the autonomous vehicle can offer a temporary remedy with regard to mobility independence. For example, there are 1% of people worldwide who suffer from epilepsy [9], a disease of neurological disorders that would make people unconscious and erupt into seizures temporarily without any forewarning. If people are driving a car when a seizure occurs, it would be a severe threat to the driver himself and related traffic participants. Therefore in most countries people with such disease are not allowed to drive. Provided there is a monitoring system on the car to predict and detect seizures of the driver, the autonomous driving system can take over the control of the car under such circumstance and thus make independent mobility a reality for those people.

With those advantages said, there is still a long way to go before autonomous vehicles with sufficient sophistication and reliability become commonly available. The available prototypes of autonomous vehicles like those introduced in [10] [11] [12] [13] [14] [15] still need a lot of tests and improvements.

### 1.1.2 From Path Planning to Motion Planning

All the advantages illustrated in Section 1.1.1 would be impossible without the help of planning. *Path planning*, *trajectory planning* and *motion planning* are three main categories of planning in this context. These items are sometimes used interchangeably in the context of robotics. They are distinguished from each other in this thesis as follows:

- *Trajectory planning* refers to generating a trajectory which is a representation of spatial positions or any other related states in terms of time.

- *Path planning* is the design of pure geometrical aspect of trajectory without considering its relationship with time.
- *Motion planning* is the calculation of a trajectory of controls.

The current planner employed on MIG is a path planner. This work is intended to upgrade the path planner to a motion planner. In the following, the reason behind this decision will be clarified after a general introduction of the current path planning system.

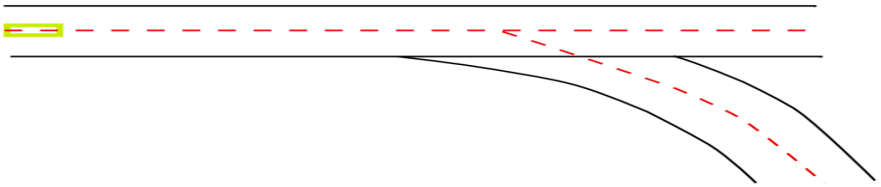
### PATH PLANNER ON MIG

One plan generated by the current path planner on MIG consists of a path spline, an edge track and an action track. The scenario reasoning modules (e.g., judgement about the right of way, traffic light detector) will generate speed recommendations for related locations on the path. Figure 1.2 shows an example as to how the planner generates a group of path splines and corresponding edge and action tracks. Given the path candidates, several cost criteria are used to evaluate each path spline. The criteria include evaluation of collision risks, lane change preference, lane preference and the remaining distance from the ending point of the path to the next check point that the car is intended to arrive at. The path with the minimum cost will be selected as the output to a path tracker which calculates the expected heading based on the pure pursuit algorithm [16]. The path tracker will also compute a desired speed according to the vehicle physics and the speed recommendations from the scenario reasoning modules. Finally, a PID controller will compute the actuation commands of steering, throttle or braking based on the desired heading and velocity and the current ones. These commands will be executed by the vehicle. The whole process is demonstrated in Figure 1.3

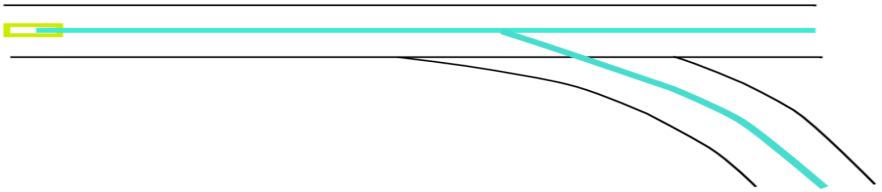
### WHY FROM A PATH PLANNER TO A MOTION PLANNER

The advantage of *trajectory planning* over *path planning* is that the former can take into account the time-related constraints and optimal criteria in a more explicit and accurate way so that it can offer better movement strategy with regard to those constraints and optimal criteria. For example, given the time information, we can evaluate the collision risks based on trajectories of the autonomous vehicle and other moving obstacles. In this way, we can know exactly whether the two objects will collide with each other or not at a point in time. For the path planner, however, we can only make a rough evaluation of the collision because we have no idea as to the location of the autonomous vehicle at a point in time in the future. The advantage of trajectory planner over path planner becomes significant in time-critical scenarios such as merging.

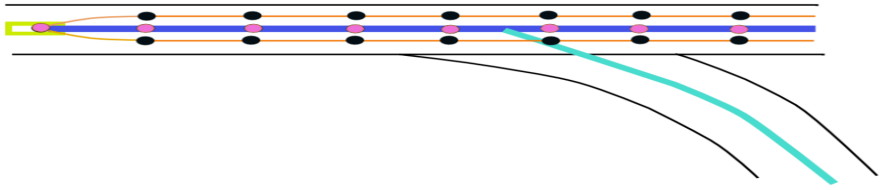
The difference between *trajectory planning* and *motion planning* lies in that an additional tracking controller is needed for the former to generate the same results as the



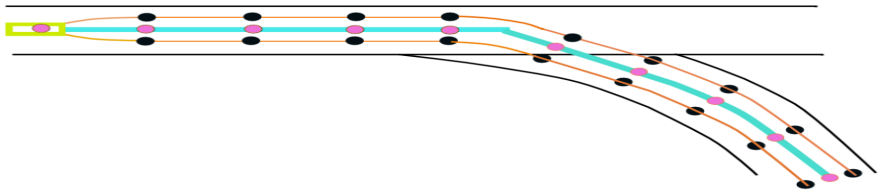
(a) The autonomous car is in front of a diverging road layout. The dashed lines are the centering lines of the roads.



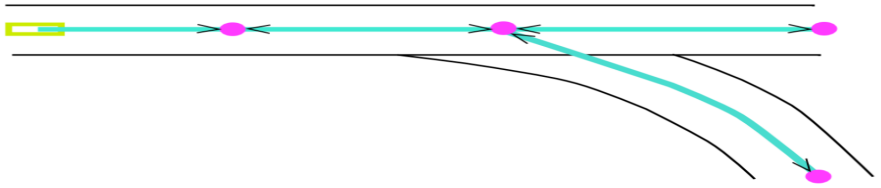
(b) Two macro plans generated by the path planner by searching the graph defined by the road network.



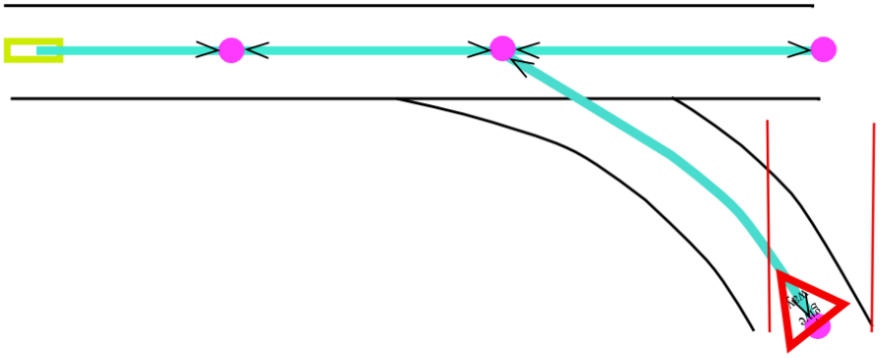
(c) Sample one macro plan and add lateral shifts to the samples to create new samples. The samples are used to generate path splines. The pink and black points are the position samples.



(d) Generate path splines for the other macro plan.



(e) The edge tracks generated by searching the graph defined by the road network.



(f) The action track will record the necessity for executing special actions before the traffic sign of give way.

Figure 1.2: An example as to how the path planner generate path splines, edge and action tracks. The green box represents the autonomous car. The area defined by the black solid lines is a segment of road network.

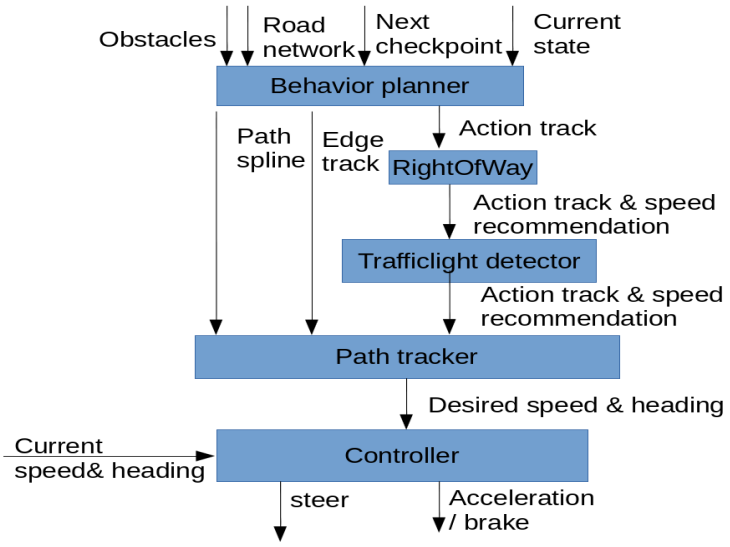



Figure 1.3: The path planning system of MIG.

latter. In other words, the controls resulted from *motion planning* can be generated by analysing the trajectory provided by the *trajectory planning*. From another point of view, the domain of *motion planning* is the control space while that of *trajectory planning* is the state space. Motion planning is adopted in this work as the author believes that it is hard to define time samples which are necessary for trajectory planning. In contrast, it is much easier to give the acceleration candidates which can be used to generate motions of the vehicle.

Due to the close relationship between *motion planning* and *trajectory planning* with respect to time, in this thesis these two terms are sometimes used interchangeably. In such circumstances, both terms are employed to represent the general idea, i.e., *planning in spatiotemporal space*. At the same time, *motion planning* and *trajectory planning* are strictly distinguished from *path planning*. Special notes would be given if it is necessary to differentiate *motion planning* and *trajectory planning* in the strict sense that follows the categorical definitions.

### 1.1.3 Application of Simulation in Autonomous Vehicle Developments

Simulation, a mathematical modelling of the physical process in real world, plays a crucial role in almost every field of research. It is also very important for autonomous on-road vehicle development. For one thing, the traffic scenarios necessary to test the system are not easily reproducible in real life. For another, the field tests can be very dangerous, costly and time-consuming.

Simulation for the testing of autonomous vehicles should consist of following components:

- car physics simulation,

- sensor simulation,
- traffic scenarios modelling ( static and dynamic objects, traffic related infrastructures, etc.).

A concrete example concerning debugging and testing the performance of motion planner can give a hint of the usefulness of simulation in this context. Initially the motion planner is implemented in a simulation environment with empty roads. Should the planner pass that simple scenario, static and dynamic traffic participants would be added to the previously empty roads. As no sensor simulation is available so far, the exact information of these traffic participants are provided for the planner to make decisions. After that, that information is replaced by simulated sensor data. Furthermore, hardware-in-loop testing can be implemented, where the car simulator is replaced by the real car while the traffic scenarios and sensor data are still simulated. In this way, the experiments conducted in a simulation environment with gradually growing realistic level save us a lot of time and effort.

In addition, simulation is also useful in terms of benchmarking and demonstration of the autonomous vehicle systems, which are important for standardization, teaching and advertising.

In terms of simulation, the main focus lies on sensor simulation in this thesis. The overall simulation framework for autonomous vehicle testing is also described generally.

## 1.2 Problem Definition of Motion Planning

The establishment of an optimal trajectory can be formulated as an optimal control problem. The process is to determine a trajectory of controls $\vec{u}(t)$ or states $\vec{x}(t)$ for a dynamic system (the car physics system) over a period of time (the planning horizon) to minimize a performance index. The performance index keeps in consistence with those optimality criteria for autonomous vehicles listed in Section 1.1.1. However it is too complicated to obtain an analytical solution for such optimal control problem due to the complicated physical and collision-related constraints [17].

Accordingly, one might consider confining $\vec{u}(t)$ to certain types of functions with unknown parameters. That is, we can let $\vec{u}(t) = f(t, \vec{a})$. In this way, the original optimal control problem is reduced to a non-linear optimization problem. However, it is still difficult to give an analytical solution due to the same problems as with the optimal control problem definition.

As a result, a further assumption that restricts $\vec{u}(t) = f(t, \vec{a})$ to a limited discrete space comes into being. The corresponding evaluations use numerical approximation of continuous constraints and optimality criteria. This is sampling approach, which is a common choice for trajectory planning for autonomous on-road driving. The main idea

is to query as many diverse trajectories as possible and select the best constraint-abiding one.

## 1.3 Evaluation Criteria of On-Road Motion Planners

The criteria for evaluating the performance of a motion planning algorithm consist mainly of *completeness*, *feasibility*, *optimality* and *runtime* [18].

### Completeness

*By definition, a motion planning algorithm is said to be complete if it can demonstrate the following property: if there is a feasible trajectory, the planner can return one in finite time; otherwise, it should report failure within a deadline [19]. Sampling based motion planning is rarely complete. It is not hard to imagine that the possibility always exists that the solution is hidden in the interstices. To that end, concepts like resolution completeness and probabilistic completeness come into being. A planner is resolution complete if it is "complete" in dealing with the discretized problem defined by itself rather than the original problem [20]. Deterministic grid or lattice based planners generally have the property of resolution completeness. Motion planners using incremental sampling like Rapidly-exploration Random Tree (RRT) may guarantee probabilistic completeness. That is, the planner is complete with probability of unity in the limit as the sampling density increases continuously [21]. Anyway, the extent to which the planner approaches completeness can be improved by sampling more densely and constructing more diversifying trajectories.*

*Completeness is crucial in situations where an emergency evasive manoeuvre is inevitable. An effective autonomous driving system should ensure that the car can find a way to avoid the collision as long as it is not in ICS. Sampling based motion planning cannot make such promise due to its incompleteness. Consequently, a fast emergency handling module is necessary for tackling critical situations. Moreover, it is helpful to have a behaviour layer that can make far-sighted decisions to avoid critical situations as early as possible. The concept of ICS-AVOID proposed in [22] highlights the usefulness of such behaviour layer.*

### Feasibility

*A trajectory is feasible if it can be tracked by the vehicle given the physics constraints. In most motion planners for car-like robots, feasibility is implicitly taken care of by the representation of the trajectory. The feasibility can be improved by increasing the*

*smoothness of the trajectory.*

### Optimality

*Given incomplete perception and highly dynamic driving environment, the motion planner for on-road driving can only search for an optimal trajectory within a limited planning horizon. Consequently the resultant solution in this way is in fact horizon-optimal rather than global optimal (if sub-optimality introduced by the discrete nature of sampling based planner is ignored).*

*Even such horizon-optimality can not be easily obtained. The first question that should be answered is what aspects should account for optimality. Another challenge lies in that different traffic scenarios might have their own emphasis on optimal driving behaviours. In [23] various cost criteria are designed and their weights are adjusted beforehand. In runtime different weights of those criteria are assigned to the motion planner by a behaviour layer according to scenario-dependant requirements. Further tests and verifications can be expected.*

*Given the concrete definition of optimality, sampling density and trajectory diversity still determine its extent.*

### Computational Complexity

*The common bottleneck for achieving a large extent of optimality and completeness is limited computational resources. Consequently most motion planners have to achieve short runtime at the expense of sacrificing optimality and completeness.*

*Besides, large runtime of the motion planner would threaten the consistency of successive plans which is crucial for stable travelling.*

So far there are in general two concrete approaches regarding sampling based motion planning. One is to deliberate multiple terminal states and construct trajectories from the ego-vehicle to each of those target states [17]. The other is to conduct graph search algorithms in a state lattice [23] [24]. Compared to the former, the latter can generate more sophisticated manoeuvres with multiple lateral motions and multiple phases of acceleration and deceleration. In this sense, the trajectory diversity of the latter is better than the former. Unfortunately, the latter renders essentially a combinatorial problem that is typically computationally expensive. In [23], a parallel algorithm is designed and implemented on the Graphics Processing Unit(GPU) to accelerate the planning process. In terms of feasibility, the trajectories constructed by the latter have insufficient smoothness.

In sum, motion planning is important for effective and safe autonomous driving and there is still large room for further improvement. In addition, given limited computational resources, an emergency handling module and a behaviour layer are so far

inevitable complements to the motion planner.

## 1.4 Thesis Contributions

The motion planner proposed in this thesis is aimed at being capable of handling complex and dynamically changing driving environments in a real time fashion. Road-adapted state lattice is employed to formulate the motion planning problem into a graph search problem. Compared with similar work presented in [23] [25][26][27], the major contribution herein is the improvement of trajectory smoothness while ensuring a high level of trajectory diversity.

Besides, another contribution of this thesis is the realization and implementation of scanning sensor(i.e., LiDAR and radar) simulation for autonomous car testing. The simulated scanning data generation is accelerated by graphics hardware via shader programming.

### Trajectory Smoothness

In [28], it is proposed that movement smoothness can be quantified as a function of jerk. By definition, jerk is the derivative of acceleration with respect to time, i.e., the third time derivative of position [29]. Smoothness (and thus jerk minimization) for trajectory planning is emphasized in several fields, especially in those of robotics and machinery processing. The objects handled by the robots can be very delicate and fragile. As a result, the jerk level of the trajectory that is followed in transferring the objects should be as small as possible. In terms of machinery processing, jerk is accountable for machine resonant vibration and premature wear-out of the machinery tools, etc.

Regarding motion planning for autonomous driving, trajectory smoothness can be beneficial in terms of the following aspects:

- It can help to provide comfortable travelling experience for the passengers.
- It can contribute to energy efficiency and environmental care. Eco-driving manoeuvres like slow acceleration and smooth driving are gaining interest worldwide [30]. In [31], simulations of vehicle travelling in an arterial corridor with traffic signals are carried out. The initial results show that in such scenarios, trajectories following smooth driving strategies can help to reduce fuel consumption and $CO_2$ emissions by around 12% in average.
- It can facilitate trajectory tracking and enable ease of control. This gain results mainly from the fact that smooth trajectories take into account implicitly unmodeled system dynamics such as unmodeled actuator behaviours [32]. This aspect is especially important for the motion planning strategy adopted in this thesis. In this approach, the planning result is a trajectory of controls rather than one of

states. Accordingly, only a low-level controller is needed while a high level tracking controller is left absent. Such low-level controller carries out the controls assigned by the planner directly without simultaneously compensating trajectory tracking errors. As a result, all the task of tracking error correction is shouldered by the motion planner itself, which is definitely much slower than a lightweight tracking controller. Consequently, a trajectory composing of smooth motions requiring a minimum amount of control effort is highly demanded.

In conclusion, trajectory smoothness is of great significance for a practical, efficient and effective motion planner. For sampling based motion planning, trajectory diversity is crucial in achieving a high level of optimality. In similar work listed at the beginning of this section, the trajectory diversity and smoothness are not achieved at the same time. This thesis proposes a trajectory representation method that can satisfy requirements from both aspects.

#### Scanning Sensor Simulation for Autonomous Vehicle Testing

One part of the thesis is focused on sensor simulation for autonomous vehicle testing. As is illustrated in Section 1.1.3, sensor simulation can introduce more realistic perception limitations in the simulation environment. Consequently, the sophistication and reliability of the motion planner can be checked and assessed more accurately. Sensor simulation can also be used to debug and validate algorithms of obstacle detection and tracking. The sensor simulation presented in this work is employed initially to assist in performance testing of a previously developed path planner and the motion planner designed in this work. After that it is also utilized in some other applications like initial design and assessment of sensor layouts for perception systems mounted on mining trucks ( cf. [33]).

There are specific requirements on sensor simulation for autonomous on-road vehicles when compared with those on traditional robot simulation platforms. As autonomous vehicles are always moving at high speed and the surrounding environment is always highly dynamic, the sensor feedback should be calculated and generated in a very frequent fashion. In this work therefore, sensor data generation algorithms are developed based on shader programming which takes advantage of the parallel computation architecture of graphics hardware. The resultant sensor simulation can update sensor data fast enough for motion planner testing.

The sensor types simulated in this work include laser scanners (Velodyne and IBEO LUX) and radars which are commonly employed in autonomous vehicle systems. For sake of convenience, these sensors are termed as *scanning sensors* in this thesis in the sense that they all send electromagnetic radiations to, and receive feedback from, the surroundings within their field of view. The video camera is also simulated to serve for

the traffic lights detector module. As the camera simulation so far is relatively simple in that it doesn't require too many special treatments with shader, it does not belong to the main focus of this thesis.

## 1.5 Thesis Structure

Chapter 2 surveys the work related to motion planning in the field of autonomous on-road vehicles. The main focus lies on motion planning based on state lattice and its related key sub-problems. Besides, the literature about LiDAR and radar simulations is also examined. Chapter 3 presents detailed illustrations of motion planning algorithms applied in planning procedures such as planning horizon specification, spatiotemporal sampling, cost map construction and state lattice construction and search. In Chapter 4, a novel trajectory representation approach is described. The trajectories generated by this approach can achieve a high level of smoothness. The application of the proposed acceleration profiles and the quality of the generated trajectories are reported. In Chapter 5, the planning strategy presented in the previous two chapters is put into practice to render an effective motion planner. The implementation of the algorithms on the CPU and GPU is presented. The cost functions applied in the motion planner are listed. The resultant motion planner is then integrated into the planning architecture of MIG. The issues with controller design, latency compensation as well as planning consistency between consecutive planning cycles are considered. So far, the description of motion planning is finished. Following is Chapter 6, where the methodology and implementation details for shader-based scanning sensor simulation are presented. Then Chapter 7 reports the performance of the proposed motion planner in several simulated on-road driving scenarios. Lastly, Chapter 8 concludes this thesis.

# Chapter 2

# Related Work

There is a large body of research regarding motion planning for autonomous vehicles. This survey focuses on motion planning based on state lattice and other types of spatiotemporal sampling. Besides, the approaches to key subproblems of state lattice-based motion planning are also reported. Those problems include spatiotemporal sampling, trajectory representation, trajectory evaluation and graph search. After that, the literature concerning scanning sensor simulations is presented.

## 2.1 On-Road Motion Planning

Many algorithms and techniques concerning autonomous on-road vehicles can be traced back to those developed for robotics research in the early days. Motion planning is no exception. [34][35][36] provide detailed illustrations and discussions about the algorithms and implementations of motion planning in the field of robotics. However, motion planning for on-road driving has its special challenges which distinguish it from its counterparts for ordinary ground robots. For example, on-road vehicles have to run in structured environments and must conform to certain traffic rules. Another instance is that the driving environment is highly dynamic and vehicle dynamics become complicated at high speed. As a result, motion planning for autonomous on-road vehicles would definitely evolve into its own way. This section reports its development until recently from DUC. Following that, the state-of-the-art techniques in dealing with its subproblems are provided.

### 2.1.1 On-Road Motion Planning in DUC

In DUC, six autonomous vehicles managed to finish the competition course. On-road motion planning then was mainly to roll out trajectories from the ego-vehicle towards several sampled target states. These trajectories were evaluated according to some criteria and the best one was selected. Such planning algorithms were devised aggressively to win the competition and only worked well in the low-density, low-speed (up to 30 mph)

contest scenarios. Nonetheless, they serve as basis for more sophisticated schemes developed afterwards that are necessary for on-road driving in complex traffic scenarios. In the following, the general framework of the planning system of the autonomous vehicle *Boss* is illustrated. Boss is from Carnegie Melon University and won the Challenge. Its planning system is very representative among autonomous vehicles and is often referred to in the literature reported in this section.

The planning framework of Boss contains mainly three separate modules (for more details, please refer to [14]):

- Mission module responsible for global route and blockage detection. It assigns a lane for the vehicle to follow so that the next checkpoint can be achieved in an efficient manner.

- Behaviour module in charge of rule-based reasoning, such as precedence judgement at intersections and lane change determination. Besides, it also performs error recovery strategies if necessary. Concretely, it assigns a lookahead distance (planning horizon) and maximum speed to the motion planner based on different scenarios.

- Motion planning module accountable for generating motion commands. Its operation is guided by the information sent from the behaviour module. It functions like this: A posture ( position and heading) is specified at the lookahead distance along the center of the assigned lane. By offsetting laterally that posture with heading retained, a column of postures are obtained. These postures then serve as short term goals for the motion planning. A curvature command spline parametrized in arc-length is employed to define the geometrical shape of a trajectory. A vehicle model is employed to check the feasibility of the trajectory. After that, several velocity profiles are applied on each path to generate a set of candidate trajectories for evaluation. These profiles include constant, linear, ramp and linear, and trapezoidal velocity types. Lastly, the motion planner feeds curvature and velocity commands to a low-level controller. For more details, see [37]. Alternatives are path planners presented in e.g.[10] and [13].

### 2.1.2 Further Improvements of On-Road Motion Planning after DUC

Based on Boss, [23] and [38] illustrate a motion planning strategy using a spatiotemporal state lattice. Several columns of postures are sampled this time rather than only one as described in [14]. Curvature is also added as an element of the posture. Two postures from different columns satisfying specific connectivity pattern would be connected to render a path edge. A curvature polynomial is still adopted in formulating the path edge, albeit updated from quadratic polynomials previously to cubic polynomials.

This adaptation is to ensure curvature continuity at the postures which might serve as switching points of an integral trajectory. Then a certain number of velocity profiles are applied on the spiral path edges to render trajectory edges. Besides constant and linear velocity profiles applied previously in [14], an acceleration profile controlled by a Proportional-Derivative (PD) law for vehicle following behaviour is also taken into account. Several trajectory edges ending at the same posture would get pruned so that only a fraction of them can have offspring. The pruning is for controlling the size of the resultant lattice so that search process can be fulfilled within a deadline. To further increase computational efficiency, parallel algorithms implemented on GPUs are designed for the planning. Various cost terms are designed and adjusted to adapt the behaviour of the robot to different traffic scenarios. The experiments in a simulation environment demonstrate effective performance of the planner in handling time critical situations where manoeuvres with multi-lateral shifts are required. Experiments on real vehicle Boss is also conducted albeit with limited driving speed and traffic scenarios of modest complexity.

A variant of the motion planner illustrated above is proposed in [25] which realizes the reduction of the computation complexity by sampling less densely but post-optimizing the resultant trajectory. The main differences and improvements of this planner compared with the one presented in [23] are as follows: The curvature cubic spiral used in [23] for the trajectory segment from ego-vehicle to the state lattice is replaced by a curvature quartic spiral. That alleviates protrusive jumps in terms of the rate of change of the curvature at the junction of two successive plans and thus the consistency of the plans is improved. Unlike the resolution-equivalent state lattice constructed on the fly as described in [23], the graph employed in [25] embodies the consideration of the temporal space by introducing a predetermined discrete speed space. The trajectory edge is thereby generated by applying a cubic velocity polynomial on the path edge. The coefficients of the velocity polynomial can be determined given the fixed velocities and an assumption of zero accelerations at the two end points of the edge. In this way, the acceleration continuity at the switch points of the trajectory is ensured, resulting in trajectories of better smoothness compared with that of [23]. However, the number of speed cells is really limited, implying less diverse trajectory profiles. Finally, the resultant trajectory is further optimized via relaxing the constraints in terms of lateral offset, curvature, heading, velocity and acceleration imposed by the fixity of graph vertices. It is reported that the performance of the resultant trajectory is improved by 10% and the computation time is said to be reduced by more than 50% in three experiment scenarios concerning lane change, static obstacles and dynamic obstacles.

[26] and [27] point out that the approach of spatiotemporal sampling and searching mentioned above wastes a lot of computation time by constructing and evaluating

trajectories that would be eventually discarded. They suggest that focused and time-consuming sampling and searching should be conducted in areas where optimal manoeuvres exist with high likelihood. In [27] [26] therefore, a two-step planning approach is proposed. The goal of the first step is to obtain a reference trajectory including a non-parametric seeding path and a regulated velocity profile that can be applied along that path. In the concrete implementation, the spatial space of interest is firstly searched and located without considering the temporal space. That implies a heuristic by which the static cost alone determines the focused spatial area. As a result, sometimes such motion planning method might fail to yield a satisfactory solution, e.g, when time-related cost, rather than static cost, should determine the focused spatial sampling space. In the second step, focused sampling is conducted in the spatiotemporal region specified by the regulated velocity profile and offsetting slightly the seeding path. The trajectory construction at this stage is similar to that of [25]. This motion planning method is reported to be practical and efficient.

The sampling patterns adopted in the motion planners presented above are based uniformly on the assumption that the geometrical shape of the trajectory is composed of curvature polynomial spirals. Allowing such assumption reduces the computational cost while lessens the diversity of the trajectories. An alternative sampling approach without confining the path type to a specific functional is applied in the motion planner presented in [24]. Along with that method, one sampled vertex represents a combination of a vehicle state and time. The vehicle state consists of a position and the first and second derivatives of the position versus time, i.e., speed and acceleration, in both lateral and longitudinal directions with regard to the center line of the road. Arbitrary vertices satisfying a specific connectivity pattern are connected via two separate polynomials parametrized in time. One polynomial encodes the lateral movement while the other the longitudinal motion. Position quintic polynomials are employed to ensure trajectory smoothness and continuity with regard to position, velocity and acceleration at the switching vertices. In this fashion, a state lattice is constructed. Exhaustive search is then implemented to obtain the trajectory with minimum cost. It is no doubt that such method must challenge the limited computational resources. Consequently in the simulation experiment the density of the sampling is very limited and the graph is constructed offline *a priori*. Parallel algorithms can help to mitigate the problem of high computation complexity, though.

Inheriting the spirit of the sampling approach illustrated in [24], [39] and [17] propose a motion planner based on a terminal manifold. The terminal manifold can be regarded as being composed of the sampled vertices resulted via the sampling method of [24]. It is called terminal manifold because all the trajectories start from the ego-vehicle and end at the sampled vertices. As no graph is constructed and thus no expensive graph-based search is necessary, the issue of expensive computation bothering [24] is avoided. In

terms of trajectory representation, the approach presented in [24] is adopted for high speed driving while the lateral movement is designed to be dependant on the longitudinal motion for low speed manoeuvres. Such treatment is argued to be reasonable. On one hand, longitudinal and lateral movements of vehicles relative to the center line of the road are highly decoupled at high speed due to the introduction of potential side slip, which favours a decoupled design of the longitudinal and lateral aspects of trajectories. On the other hand, no side slip is a valid assumption at low speed and thus the nonholonomic constraint plays a key role, which would invalidate most of the trajectories due to invalid curvatures if the decoupled design is followed. It is shown in simulation environment that several challenging traffic scenarios in both urban and highway traffic are handled properly by the resultant motion planner.

### 2.1.3 Approaches to Subproblems of Lattice-based Motion Planning

This section focuses on four subproblems which are essential for state lattice-based on-road motion planning, that is, spatiotemporal sampling, trajectory representation, trajectory evaluation and graph search. Motion planning is an important topic in the field of robotics. Several subproblems of on-road motion planning for autonomous vehicles can be solved by adapting the solutions for the similar problems established in motion planning for mobile robots, especially car-like robots. The illustration of this section takes care of this aspect.

#### 2.1.3.1 Spatiotemporal Sampling

As it is hard to obtain an accurate analytical solution for the on-road motion planning problem, approximation is necessary for calculating a solution within a time deadline [18]. Spatiotemporal sampling is nothing but an approximation of the continuous spatiotemporal space. In earlier days of the autonomous vehicle development, sampling for on-road motion planning was limited to generating several targets (states or configurations) for the planning in one planning horizon. As is mentioned previously in Section 2.1.1, trajectories (or paths) constrained to a specific functional are then constructed to connect the ego-vehicle and the targets. One way to diversify the trajectory expression rather than restricting it to a specific functional is to sample intermediate states as well as the terminal targets. The trajectory primitives connecting two states then serve as one part of the integral trajectory. In this fashion, more expressive trajectory types can be obtained such as those with multi-phases of accelerations and multi-shifts. In fact, the trajectory primitives can also be regarded as one part of the sampling. Such spatiotemporal sampling results in a graph with vertices being the sampled states and the edges being the trajectory primitives. Such graph can also be called a *state lattice* in the sense that each vertex encodes the state information of the vehicle and a specific connectivity pattern is applied in the edge construction.

Deterministic and random samplings are the two basic categories of sampling methods. Both of them have been applied in on-road motion planning.

The motion planners described in Section 2.1.2 uniformly adopt deterministic sampling. The basic guideline in sampling for on-road planning in terms of spatial space is that the sampling should be adapted to the road shape. That is crucial for planning efficiency and generation of human-like behaviours. Normally the center line of the road is set as the basis of the sampling and the vertices are generated by offsetting laterally the postures sampled along the center line. The resultant spatial vertices can be augmented by additional temporal elements such as time [24][23], speed [25][26] [27] and acceleration [23] [24]. The representative value of the vertex can be predetermined [24][25][26][27] or be calculated on the fly [23]. For detailed accounting, please refer to Section 2.1.2.

RRT is among the most important random sampling algorithms with respect to motion planning for car-like robots. RRT is a tree of dynamically feasible trajectories constructed on the fly by the guidance of a simple stochastic strategy [40]. It can quickly explore the free sampling space. Figure 2.1 demonstrates an RRT constructed for on-road motion planning. At the beginning of each iteration, a random state is firstly sampled in the free space. Then a node on the tree is selected to be the start state according to some heuristic. Such heuristic is generally called Nearest Neighbour (NN) metric. After that, a trajectory is propagated by simulating forward the dynamic system from the start state towards the sampled state. Finally, the resultant end state and the trajectory would be added to RRT as a new node and a new edge respectively if the trajectory survives collision checking [40]. This procedure repeats until certain sampling metric is satisfied or specific time limit is met. This data structure can be easily adapted to motion planning problems with different dimension combinations and various dynamic models because only a forward dynamic model is needed [24]. [41][42] demonstrate the navigation potential of RRT by employing it in motion planning for autonomous vehicles.

The main advantage of RRT over deterministic sampling is that the former does not suffer from dimension curse, since the number of nodes on the tree does not grow exponentially with respect to the dimension (as they are randomly sampled on the fly). With careful design, it can adapt to various search space easily and motion planners based on it can run efficiently.

Another merit of RRT is that it has certain completeness guarantees[42], i.e., probabilistic completeness. The planner can continue exploring the free space until a solution is located. However, to what extent that virtue can play a role in handling critical situations such as collision avoidance is largely determined by the runtime it needs to find that solution. That is hardly determined due to its stochastic nature.

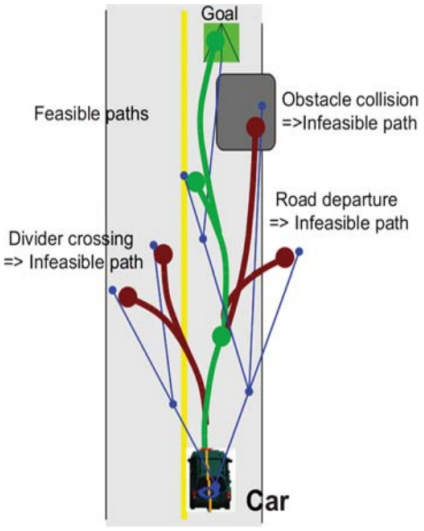



Figure 2.1: An RRT generated by the RRT-based motion planner implemented in MIT's Talos, from [43].

One drawback of RRT in general is the fact that its efficiency and the optimality of the generated trajectory are very sensitive to the choice of NN-metric and it yields good result only if the NN-metric approximates the real travel cost properly [44]. Normally applied NN-metric is Euclidean metric [45]. In [42], Dubins path length is employed as the NN-metric. Anyway, it is hard to devise a heuristic that is a good approximation of the real travel cost if e.g., energy efficiency, comfort, vehicle physics constraints are taken into account.

#### 2.1.3.2 Trajectory Representation

Trajectory representation can be formulated as an optimal control problem with boundary conditions being given start and end states. However, it is hard to get a general analytical solution for that optimal control problem due to potential varying constraints. As a result, some parametric functional is usually presumed, which converts the optimal control problem to a parameter finding or constrained optimization problem if there are more than one feasible candidates [46].

In the context of motion planning for car-like robots, there are basically two approaches concerning parametric trajectory representation. One confines the geometrical shape of the trajectory (i.e., path) to specific parametric functional. The trajectory would then be generated by applying some velocity profiles on the path. The other has no assumption about the path type of the trajectory; instead, the trajectory of the vehicle is regarded as a combination of two elementary trajectories in terms of longitudinal and lateral references respectively.

There is a detailed survey with respect to the methods of path representation in [47][48] [46]. In general, there are several methods such as straight lines [49], circular arcs

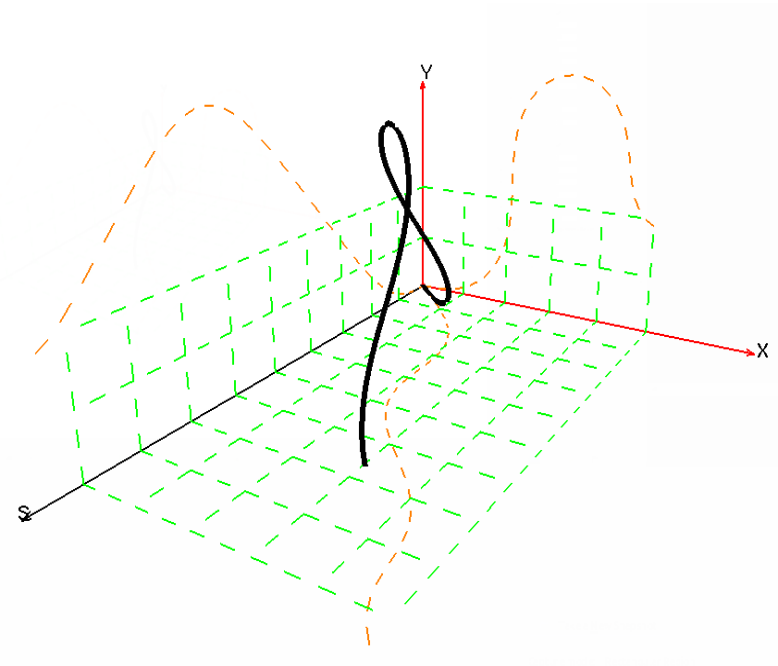



Figure 2.2: Cubic spiral. The curvature of this curve is expressed in a cubic polynomial in the form of $\kappa(s) = p_0 + p_1 s + p_2 s^2 + p_3 s^3$. In this example, the coefficients are $p_0 = 0, p_1 = 0.228, p_2 = -0.045, p_3 = 0.0019$.

[50] , clothoids [48], Bezier curves [47], cubic spirals(curvature quadratic) [48], Akima splines [51], curvature cubic polynomials [46] [23] , curvature quartic polynomials [25], etc. These path primitives have different orders of curvature continuity at the switching points of consecutive path segments. In the motion planners designed in [23] [27] [25] a path edge is generated based on the approach of path representation proposed in [46]. Figure 2.2 shows a cubic spiral which is a curve resulting from a curvature input expressed in a cubic polynomial.

As for the choice of velocity profiles, Section 2.1.2 gives some details on that along with the illustration of the motion planners. In Chapter 4 more detailed illustrations as well as comparisons on that topic are provided.

Regarding the longitudinal-lateral approach, [52] and [17] formulate the trajectory representation problem as two independent optimal control problems, each for one reference. The performance index of the optimal control problem is smoothness, embodied by minimum jerk concretely. Without consideration of constraints, they prove that the position quintic polynomial parametrized in time is the solution. In [17], a motion planner based on such trajectory representation is implemented for on-road autonomous driving. The center line of the road is chosen to be the lateral reference while the perpendicular direction with regard to the center line is set as the lateral reference. Those separately generated longitudinal and lateral trajectories are combined and analysed afterwards to calculate vehicle motions such as curvatures and accelerations. That information is used

for physics constraints checking and for tracking control.

#### 2.1.3.3 Trajectory Evaluation

Trajectories are always further sampled and each sample is evaluated according to some cost function. If there are many trajectories to be evaluated, it is necessary to construct a *cost map* as a lookup table for efficient queries of cost values. A cost map is a sampled version of the cost function. Defined in concrete terms, it is a collection of cells with each having a representative state for which the cost function is evaluated [18]. In this way, the evaluation of a trajectory is reduced to a summation operation of its occupied cells in the cost map. A set of such cells is called a *swath* of the trajectory [18].

Trajectory evaluation including collision checking and motion and state cost estimation, as is generally acknowledged in the field of robotics, is very time-consuming [18] [53] [54] [55]. It accounts for a large part of the computation time for motion planning. That is true especially in cases of exhaustive search-based planner where a large amount of trajectories need to be evaluated [56]. To mitigate that challenge, *cost map* is widely used in robotics for trajectory evaluation. Although the construction of cost map is relatively expensive, that effort gets paid off easily in applications where tens of thousands of evaluations are needed [56].

The following focuses on the issues involving collision-related cost maps. An *original cost map* is defined to be a cost map constructed based on the original obstacles, i.e., obstacles without any form of dilation around them.

A trajectory describes the motion locus of a representative point on the vehicle. If the shape of the vehicle cannot be ignorable compared to the representative point, it must be taken into account when it comes to collision checking. Consequently, given an original cost map, the evaluation is conducted by convolving the vehicle frame with the cost map along the trajectory in question. Figure 2.3 gives a demonstration of the convolution. As such convolution is still expensive, it is much better to dilate the original cost map in such a way that the compensation for the vehicle frame is implicitly accounted for in the resultant cost map. In this way, it is safe enough to just evaluate the swath of the trajectory, which is much more efficient.

For orientally invariant disk shape or the kind of shapes that can be approximated by disk shapes with tolerable discrepancy, the dilated cost map can be constructed by expanding the occupied cells in the original cost map by the radius of the disk shape. For car-like robot, however, its rectangular shape with an aspect ratio of approximately 5 to 2 [56] requires that its orientation has to be accounted for. Figure 2.4 gives a hint as to how the orientation of the ego-vehicle affects the dilation of the obstacles. This particular issue with car-like robot makes the collision checking not so easy as that of the disk shape robot.

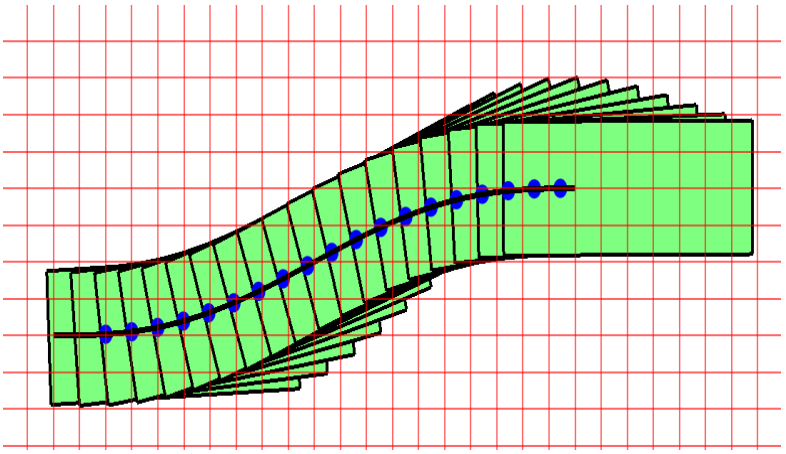


Figure 2.3: Convolution of a vehicle frame with a cost map. The green box represents the vehicle frame. The blue point refers to the representative point on the vehicle frame. The black curve is the trajectory recording the locus of the representative point. The cost map is defined over the red grid.

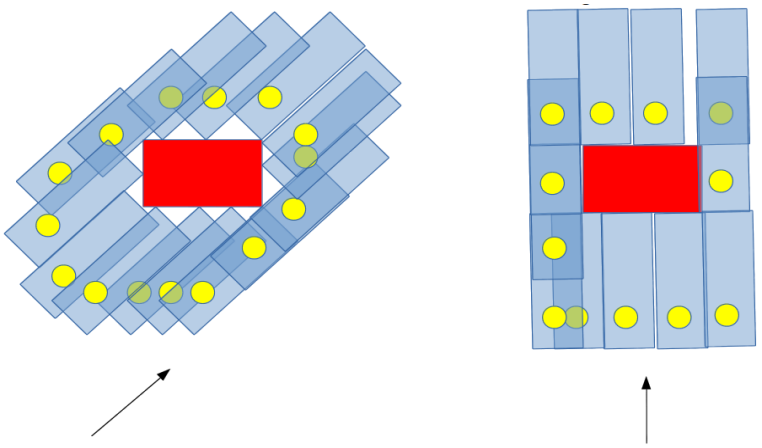


Figure 2.4: The dilation (by connecting the yellow points) of the obstacle(red) according to different orientations of the vehicle(blue).

In [57] a 2D grid-based cost map is designed for efficient collision checking and cost evaluation. Such cost maps are generously applied in the state lattice planner for autonomous driving presented in [55]. They expand the original cost map by radius of an inscribing circle within the vehicle frame and radius of a minimum circle bounding the vehicle, which results in two different cost maps respectively, i.e., *optimistic cost map* and *pessimistic cost map*. Assuming a point robot, if any cell on the *optimistic map* along the trajectory is non-free, then the trajectory is guaranteed to be untraversable; if all the cells on the *pessimistic cost map* along the trajectories are free, then it is concluded that the trajectory is traversable. In this way, only the trajectories that fail both tests (i.e., the cells on the *optimistic cost map* covered by the trajectory are all free while some of the cells on the *pessimistic* cost map occupied by the trajectory are non-free) have to be convolved with the cost map using the full-scale vehicle shape for

further detailed checking. The computation is further reduced by calculating the cells covered by the full-scale vehicle executing specific trajectory offline *a priori* so that the online evaluation is reduced to a summation operation over such set of cells.

In [56], the rectangular vehicle frame is represented by several overlapped disk shaped primitives. Each disk shape is further decomposed into several axis-aligned rectangles with axis conforming to the x and y coordinates of the original discrete cost map. That operation is feasible thanks to the nature of the disk shape in terms of invariant orientation. In this way, the dilation of the cost map can be broken down into several simple and efficient dilation procedures.

In [23] it assumes that in scenarios of highway driving, the heading deviation of autonomous vehicles from parallel to the center line of the road is within six degrees. That assumption in turn specifies the size of dilation with respect to road-adapted coordinates.

#### 2.1.3.4 Graph Search

If a graph, or rather, a state lattice, is constructed for motion planning as described in [23] [38][25][26] [27][24], efficient graph search algorithms should be implemented to find the best constraint-abiding trajectory.

There are many graph search algorithms available in the literature, such as A$^*$, D$^*$[58], focused D$^*$ [59], anytime dynamic A$^*$ [60], etc. However, proper heuristics for estimating future trajectory cost are necessary for those algorithms mentioned above to be efficient and to yield optimal solutions [61]. Similar to the situation regarding RRT as mentioned in Section 2.1.3.1, it is difficult to find a heuristic which well encodes the many cost criteria against the trajectories for autonomous on-road driving. Besides, those algorithms require that the target state of the path or trajectory is given. That is normally not the case for on-road motion planning where all the sampled vertices may be regarded as a potential target. Consequently the fixed-target graph search algorithms have to be implemented for each of the sampled targets, which makes them inefficient in a global sense. As a result, exhaustive algorithms [24] or dynamic programming-based search algorithms [23][25][26] are always the choices for search-based on-road motion planning.

## 2.2 Scanning Sensor Simulation

The simulation of scanning sensors, namely LiDAR and Radar, mainly contains three aspects, i.e., definition of sensor specification, environment modelling and calculation of interactions between the scanning rays (or waves) and the environment. The first factor varies across sensors and it is not a focus of this thesis. Accordingly, the following report emphasizes the last two aspects. Besides, computation approaches are also recorded if the corresponding information is available.

It is noteworthy that simulation of the transmissions of scanning rays or waves with a high realistic level can be realized by employing the ray tracing technique [62]. However, modelling based on ray tracing is too computation intensive [63]. That is because it requires essentially taking into account all the reactions between the light (ray or wave in the context of scanning sensors) and the objects encountered by it during the transmission [64]. There are efforts to improve the computational efficiency of ray tracing, such as implementing parallel ray tracing algorithms on graphics hardware [65] and even designing specific hardware named Ray Processing Unit in competition with existing GPUs [66]. Nonetheless, such techniques are still not ready for common and easy access. As a result, so far most scanning sensor simulations for real time applications employ simplified versions of the complicated physical process to different degrees.

### 2.2.1 Airbone Scanning Sensor Simulation

[67] presents airborne imaging radar simulation applied in flight simulators for man-in-the-loop training of pilots and radar operators in Air-to-Ground (A/G) mode of operation. Environment models consist of terrain and weather. The terrain is described by 2D grid topography with each cell containing information of radar reflectivity and surface attributes. The weather is represented by a 3D grid of rainfall rates. The sensor's physical model includes detailed simulations of antenna, transmitter and receiver which belong to the main components of a radar device. The radio waves are modelled by sampling discrete range traces at closely spaced azimuth angles and generating range samples for each range trace based on all the data points between neighbouring range traces. In that way the gaming area is determined. Then the signal strength is calculated based on radar physical equations in terms of various media and targets encountered by the radio wave during its transmission. After being further processed by the antenna and receiver modules, the resultant signals are used to generate a radar view. The simulation is mainly conducted on a graphics workstation provided by Silicon Graphics Inc.(SGI).

In [62] millimetre wave (MMW) imaging radar simulation is constructed for study of the effect of MMW Radar on assisting pilot in approaching, landing and taxiing under bad weather conditions. 3D database of airports and their neighbourhood augmented with simulation related microwave properties is employed as the environment model. The main idea is to make use of depth and color images in the radar simulation. Those images are generated naturally along with visual rendering via rendering pipeline standards of Open Graphics Library (OpenGL) . Additional transformations are carried out in CPU. The final information produced thereof is that of the target locations and the intensity of the reflected signals. The simulation runs on an SGI graphics workstation.

[68] manipulates techniques of vertex and fragment shader programming to realize the simulation functions of the previous work presented in [62]. Shader programming is

supported by GPUs compatible with the programmable OpenGL rendering pipeline. It is no longer necessary for the CPU to read the image out from the GPU as no additional transformations are necessary to perform in the CPU. As all the calculations can be conducted on the GPU, computational efficiency is increased.

Using computation approaches similar to [68], [69] presents a LiDAR simulation for testing and evaluating pilot assisting multi-sensor vision systems.

### 2.2.2 Scanning Sensor Simulation in the Field of Robotics

In general, sensor simulation is common in simulators for robotics applications. One example worth to mention is the Player project[70]. It is an open source project consisting of three components, i.e., Player, Stage and Gazebo. Player provides an abstracted interface between the control system of the robot and the many kinds of hardware, be it sensor or actuator. Stage and gazebo can simulate a population of mobile robots, sensors and scenario related objects. The difference between Stage and Gazebo lies in that Stage simulates multi-agent in a 2D bitmapped environment whereas Gazebo in a 3D environment. The sensor types provided by Stage and Gazebo include range-finders (sonar, SICK and Hokuyo laser scanners, IR), vision (color blob detection), 3D depth-map camera, odometry (with drift error model) [71]. One of the most popular robotics framework, Robot Operating System (ROS), also enables interfacing with the Gazebo simulator. Two different approaches are implemented in Gazebo to simulate ray sensors. One uses the collision engine contained in third-party software Open Dynamics Engine which is an open source physics engine. The other employs the rendering interfaces provided by Open source Graphics Rendering Engine (OGRE) which is a 3D graphics engine [70].

As far as sensor simulations in the area of autonomous vehicles are concerned, there are already some commercial products available. Two examples are PreScan [72] from TassInternational Company and Traffic Simulation (TRS) from Nisys Company [73]. Both of them provide simulations of sensors and traffic scenarios combined. Sensor types available in PreScan are radar, LiDAR, camera and GPS while those of TRS include video, radar and CAN.

GPU supported methods such as those using depth images and OpenGL shaders are argued to be computationally efficient for simulations of time-of-flight sensors for intelligent vehicle systems in [63]. A simple demonstration is presented without further algorithm and implementation details.

In [74] a shader-based LiDAR simulation is practised in a simulation framework for autonomous vehicles. That is however too naive to be be applied in simulations for autonomous vehicle testing.

# Chapter 3

# On-Road Motion Planner Design

The core idea of the proposed planner is to generate numerous diversifying trajectories within one planning horizon and select the best constraint-abiding one. This scheme is realized by searching in a lane-adapted state lattice. The state lattice is constructed via deterministic sampling in the spatiotemporal space. In order to make the resultant problem computationally tractable, a parallel algorithm is devised to carry out the expensive graph construction and search. This chapter illustrates main algorithms concerning this planning strategy. Besides, assumptions, design guidelines and analyses are also provided where necessary.

## 3.1 Specifying the Spatial Horizon

It should be first pointed out that in this section, the term *spatial horizon* refers particularly to the road segments of interest within a given travel distance $d_H$. In other words, $d_H$, which is also an aspect of the spatial horizon in the general sense, is assumed given and no longer needs to be specified.

The issues concerning the spatial horizon are often omitted in the literature related to on-road motion planning. That is because it is usually assumed that the mission or behaviour layer is accountable for assigning a road to the motion planner. Besides, the roads in the test environments are often chosen in such a way that they have only simple structures. As the motion planner designed in this work is intended to operate smoothly and consistently in road networks with various layouts, a general approach to specify the spatial horizon that can adapt to different road layouts is necessary.

### 3.1.1 Assumptions

Several assumptions about the autonomous vehicle system as well as the real road network have to be made as they are necessary for the presented strategy to be implemented effectively.

The method is devised based on the road network model (named as an *RNDF* graph) applied in the autonomous driving system of MIG. Accordingly, the first assumption is

that the *RNDF* graph for the real road network is given. Besides, the assumptions for building the *RNDF* graph are inevitably inherited by the proposed spatial-horizon specification approach. Two important assumptions among them are: 1) a uniform lane width along one separate lane segment, 2) parallel left and right lane boundaries of equal distance from the center of the lane. The following provides some information and terminologies regarding the road network model, which are necessary for understanding this section. For more details, please refer to [75] [76].

#### ROUTE NETWORK DEFINITION FILE (RNDF)

The *RNDF* is issued for DUC. It classifies the driving environments into two types, i.e., *road segments* and *zones* (e.g., parking lots). A road segment may contain several neighbouring lanes. A lane is characterized by the nominal lane width and the ordered set of *waypoints* along its center line. A waypoint contains the information of its location and can be associated to the connections between lanes, traffic lights, stop signs, etc. Waypoints that the vehicle is required to reach are denoted as *checkpoints*.

#### RNDF GRAPH

In the software system of MIG, the *RNDF* graph is built based on the given RNDF to communicate with the planning modules. The edges of the *RNDF* graph may represent the center lines of

- actual lanes,
- virtual lanes added to facilitate the behaviour of lane change,
- and virtual lanes added as mirrored lanes of real lanes to facilitate manoeuvres like swerving and overtaking via neighbouring lanes with oncoming traffic.

The nodes of the *RNDF* graph can be either actual waypoints connecting adjacent lane segments or virtual waypoints added for defining the virtual lanes. The center line of a lane segment is modelled as a spline with supporting points being the waypoints lying on it. A lane may contain several edges, and each edge records the information of its neighbouring lanes. Whether arbitrary two lanes are neighbours is determined by a statistical estimation. Figure 3.1 demonstrates an example *RNDF* graph.

### 3.1.2 Definition and Construction Principles

It is intuitive to incorporate all the road segments within a given travel distance in front of the vehicle into the spatial horizon. Given a portion of the road network as shown in Figure 3.2(a), the spatial horizon specified in this way is demonstrated in Figure 3.2(b). However, such spatial horizon is so large that it may entail prohibitively expensive planning. To that end, Principle 3.1 for defining the spatial horizon is introduced.

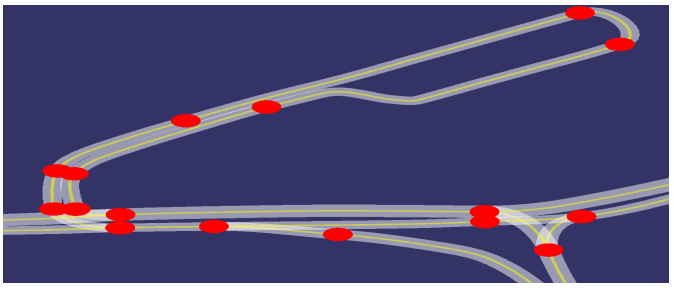


Figure 3.1: An example *RNDF* graph. The yellow curves are the spline edges. The red points are the nodes. The white bands represent the actual road segments.

**Principle 3.1.** *The long-term plan should not be affected by the cost of short-term goals, unless the cost of the latter becomes infinite.*

Principle 3.1 is in fact followed by the mission or behaviour module of most autonomous vehicle systems when they assign lanes to motion planners. Concretely, it favours road segments that would lead to a shorter path to the next mission goal. It does so even if the travel costs of the trajectories within the planning horizon on those segments are larger than on others. Only when the favourite road segments are untraversable due to, e.g., blockage, the other roads would be considered. From the perspective of the performance of motion planners, Principle 3.1 implies trading optimality for computational efficiency.

Figure 3.2(c) displays the spatial horizon modified according to Principle 3.1 based on the one shown in Figure 3.2(b).

To facilitate sampling and graph construction on the resultant spatial horizon, the next principle is brought about:

**Principle 3.2.** *Two separate segments of the spatial horizon, when projected onto the lateral reference (any of the center lines of the roads containing the segments in question), should not overlap. Every segment should have a uniform width itself.*

Figure 3.2(d) demonstrates the updated spatial horizon after Principle 3.2 is applied.

It is argued in [17] that successive plans should keep *temporal consistency* (TC). TC is defined in the context of motion planning in this thesis as follows:

**Definition 3.1.** *Temporal consistency: The remaining segment of the best trajectory generated from one planning cycle can still be contained in at least one candidate trajectory in subsequent planning cycles, under the assumption that there are no computation and tracking errors. However, whether the trajectory responsible for the consistency can survive the checking of constraints and be selected as the best one in the following planning cycles will still be subject to the new information introduced into the world model.*

Correspondingly, another principle is proposed to stress TC from the perspective of defining the spatial horizon:

**Principle 3.3.** *If the point $(x, y)$ is contained in the spatial horizon at one time, it will still be enclosed in subsequent spatial horizons until it no longer lies in front of the vehicle.*

It is noteworthy that motion planners with the property of TC do not necessarily follow Principle 3.3. For example, they can just define the spatial horizon to be an area where the remaining trajectory lies. Nonetheless, Principle 3.3 ensures a sufficient spatial horizon for motion planners with the requirement of TC.

Taking into account the aforementioned principles for the definition of the spatial horizon results in the following principles for its construction:

**Principle 3.4.** *All the road segments in front of the vehicle within a given travel distance should first be gathered. Then unnecessary segments are removed from the originally complete collection. Whether a segment is necessary or not is determined based on Principle 3.1.*

**Principle 3.5.** *The neighbouring road segments should be integrated into one horizon segment. If the width within the resultant segment is not uniform, this segment should be further divided to into smaller segments to guarantee Principle 3.2.*

It should be pointed out that the proposed principles for specifying the spatial horizon trade computational efficiency for planning consistency and flexibility to some extent. For example, in practice, the chance that the vehicle might use the short neighbouring road segment shown in Figure 3.2(c) is rather slim. Additional treatments are still necessary, e.g., to delete that neighbouring segment when it gets undesirably short. This issue is left for future work.

### 3.1.3 Data Structures and Algorithm for Spatial Horizon Construction

DATA STRUCTURE

- The *LATTICE-SEC* encodes the information of one segment that is constructed in accord with Principle 3.5. Its main elements include the segment's width, longitudinal distance, lateral reference representation, the layout of spatial nodes, and its relationship with the horizon segment next to it. If the *LATTICE-SEC* consists of several neighbouring lanes, the nearest one to the vehicle is selected to provide the lateral reference for the entire segment.
- The *LATTICE-SEC-ARRAY* is an array of *LATTICE-SEC*s and satisfies Principle 3.4. It has the complete information of the sampled spatial horizon and can

(a) A portion of the road network where the spatial horizon is specified.

(b) Spatial horizon containing all road segments in front of the vehicle within a given travel distance.

(c) Spatial horizon defined on road segments that lead to a shorter path to the next mission goal.

(d) Spatial horizon with segments of uniform characteristics.

Figure 3.2: Specification of the spatial horizon under the guidance of principles for the definition of the spatial horizon. Road segments with different colors are supposed to have different spline representations from the perspective of the planner.

be directly applied for graph construction. The *LATTICE-SEC*s are organized in ascending order, based on their longitudinal distance to the start of the spatial horizon.

- The *LATTICE-SEC-TREE* and the *LATTICE-SEC-ARRAY* are different in that the former is in the form of a tree structure rather than an array.
- The *LATTICE-SEC-RAW* is the forerunner of one or several *LATTICE-SEC*s. Its width might not be uniform.
- The *LATTICE-SEC-RAW-TREE* is a tree structure with several *LATTICE-SEC-RAW*s as its nodes.

CONSTRUCTION ALGORITHM

- The *ROUGH-CONSTRUCTION* follows Principle 3.4 albeit without the pruning procedure. At the same time, the first part of Principle 3.5 is taken into account. Concretely, the breadth-first search is conducted on the *RNDF* graph first in order to locate the neighbouring lanes of the current lane (including the current lane itself). Then the depth-first search is performed to trace forward each of the lanes discovered in the breadth-first search until it reaches a given travel distance. During the search, a *LATTICE-SEC-RAW* would be constructed any time to record the information of an edge that is represented by a spline different from the one representing its precedence along the same lane. Moreover, the existing *LATTICE-SEC-RAW*s would get updated to incorporate the information of newly located neighbouring or following lane segments. The overall process yields the *LATTICE-SEC-RAW-TREE* as its final result.
- The *FOCUSED-CONSTRUCTION-AND-SAMPLING* applies the second part of Principle 3.5 on each *LATTICE-SEC-RAW* of the *LATTICE-SEC-RAW-TREE* generated from the last procedure. This procedure also defines the layout of the spatial nodes but does not calculate their concrete representative values. Its result is a *LATTICE-SEC-TREE*.
- The *PRUNING* removes the undesired sections of the *LATTICE-SEC-TREE* according to Principle 3.4, and its result is a *LATTICE-SEC-ARRAY*.

Figure 3.3 displays a typical example of the construction process. One might have the impression that it would be more convenient if the *PRUNING* process could take place before the procedure of *FOCUSED-CONSTRUCTION-AND-SAMPLING*. The consideration that leads to the current algorithm is twofold. On one hand, the *FOCUSED-CONSTRUCTION-AND-SAMPLING* is negligibly lightweight so that the processing

of some ought-to-be-pruned segments does not impair computational efficiency at all. On the other hand, the layout of the spatial nodes resulting from the *FOCUSED-CONSTRUCTION-AND-SAMPLING* procedure can facilitate blockage checking which is necessary for observing Principle 3.4. In this way, if the lane segments supported by the long term goal turn out to be untraversable, the other segments are ready to be verified and utilized if necessary. As a result, the *PRUNING* serves as the last procedure of the spatial horizon construction in the proposed planner.

## 3.2 Spatial Sampling

Given the planning horizon, the next step is to perform sampling. In the proposed motion planner, the spatial space and the temporal one are sampled separately. The result of the spatial sampling is the spatial graph $\{\hat{n}, \hat{e}\}$ ( $\hat{n}$ and $\hat{e}$ represent the spatial node and the path edge respectively). It is noteworthy that the accents above the notations are adopted to distinguish the spatial graph from the spatiotemporal graph (also called state lattice, denoted as $\{n, e\}$ ) which is the final output of the complete sampling. Spatial sampling is focused in this section, while temporal sampling is the main topic of the following section.

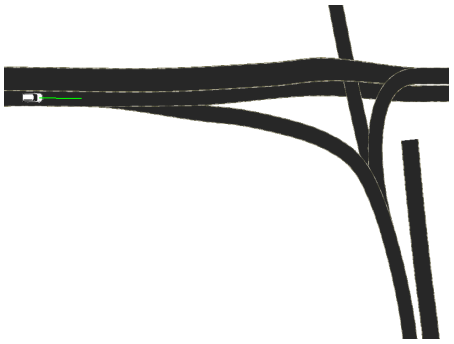
### 3.2.1 A Lane-Adapted Coordinate System

A straightforward approach to sample the spatial space is to construct a uniform grid in $X - Y$ space. That approach is feasible for sampling straight lanes. When it comes to other road shapes (e.g., a curve), however, the uniform grid would become troublesome. As can be seen from Figure 3.4, it is hard to implement one single uniform grid to cover the entire road surface without having to waste efforts in unnecessary areas. For more efficient sampling, a lane-adapted coordinate system (termed as $SL$ coordinate system) is introduced in [23] [24]. As is shown in Figure 3.5, the axis $S$ coincides with the center of the lane, and $L$ can be any line that is perpendicular to $S$ within the planning horizon. In this way, the $s$ coordinate of a point $(s, l)$ in $SL$ frame refers to the arc length along the center line from the start of the horizon to the projected point of $(s, l)$ on $S$. $s$ is also called *station*. The $l$ component of $(s, l)$ denotes the lateral offset of the point from the center line.

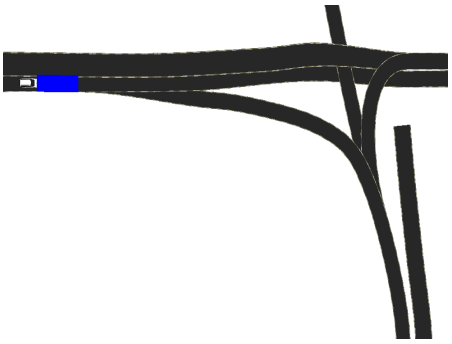
### 3.2.2 Spatial Node

The spatial nodes serve as the endpoints of the path primitives. Given the $SL$ space as the domain and a right-handed coordinate system, the spatial node is represented as:

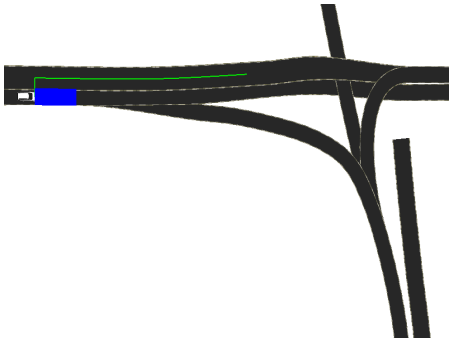
$$\vec{\gamma}(s, l) = (x_n(s, l), y_n(s, l), \theta_n(s, l), \kappa_n(s, l))^T \tag{3.1}$$



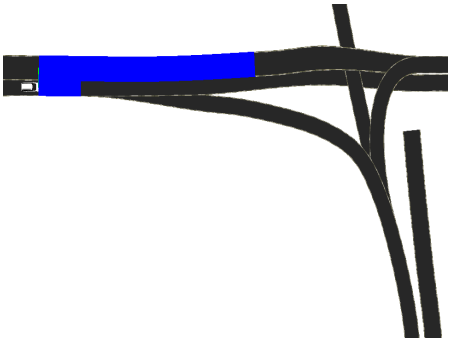
(a) An edge along the current lane is found.



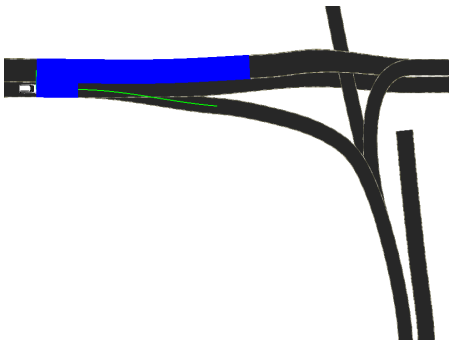
(b) A *LATTICE-SEC-RAW* (blue) is constructed to record the located edge.



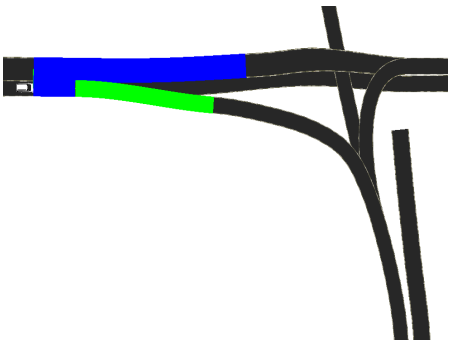
(c) Another edge is found on the neighouring lane.



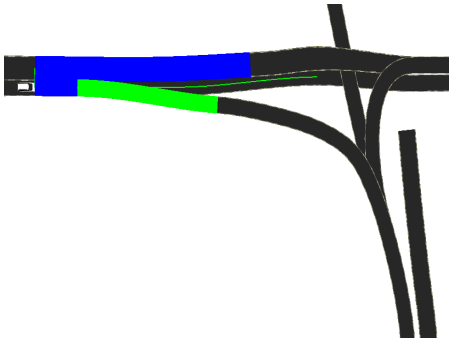
(d) Neighbouring lanes are encouraged to be combined with each other, so the information of the newly located edge is again recorded in *LATTICE-SEC-RAW* (blue). *LATTICE-SEC-RAW* (blue) gets updated accordingly.



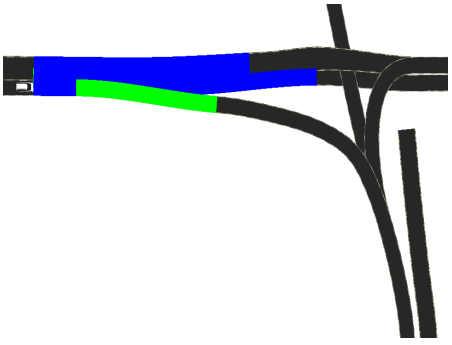
(e) Another edge is located.



(f) As the newly discovered edge is represented in a different spline compared with its parent edge, a new *LATTICE-SEC-RAW* (green) is created to log it.

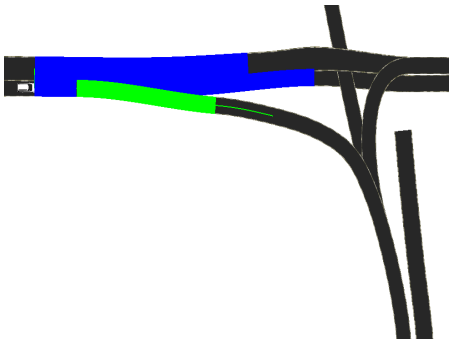


(g) Another edge is discovered.

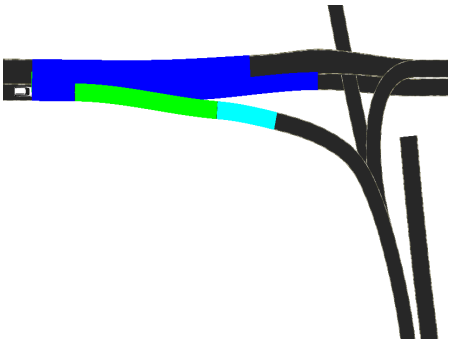


(h) As the newly located edge is in the same spline as its parent edge, it is recorded in the same *LATTICE-SEC-RAW* as its parent edge. In this way, *LATTICE-SEC-RAW* (blue) is updated.

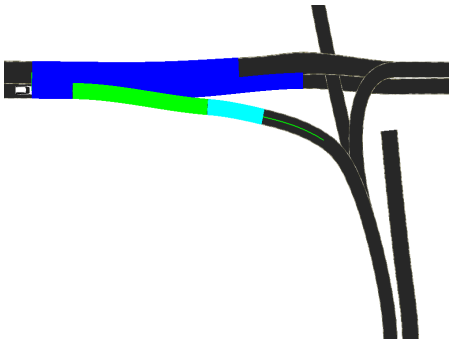
Figure 3.3: An example of the construction process of the spatial horizon.



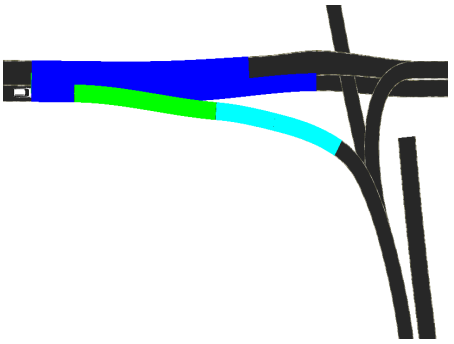
(i) Another edge is uncovered.



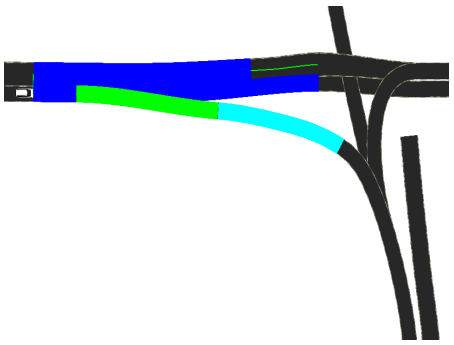
(j) As the newly discovered edge and its parent edge are represented in different splines, a new *LATTICE-SEC-RAW* (cyan) is created to record the former.



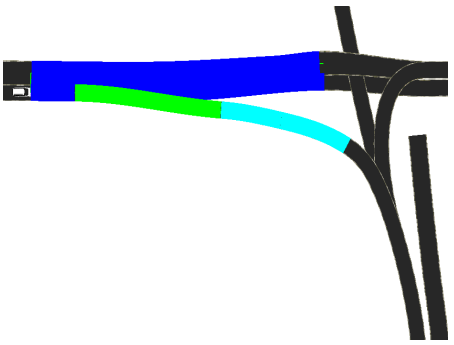
(k) Another edge is discovered.



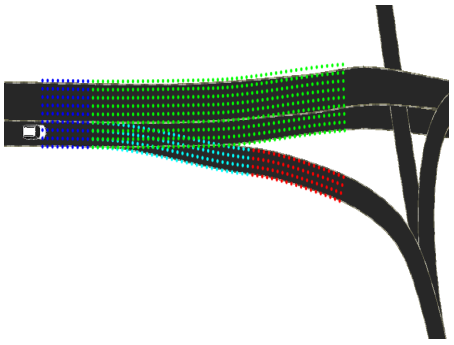
(l) As the newly discovered edge and its parent edge are represented in the same spline, *LATTICE-SEC-RAW* (cyan) is updated to record the former.



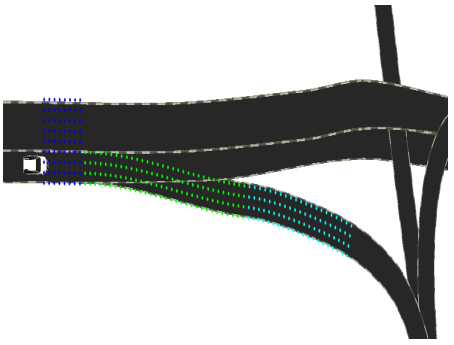
(m) Another edge is discovered on the neighbouring lane.



(n) As the newly discovered edge and its parent edge are represented in the same spline, *LATTICE-SEC-RAW* (blue) is updated to record the former. So far, the road segments in front of the vehcle within a given distance are all recorded.

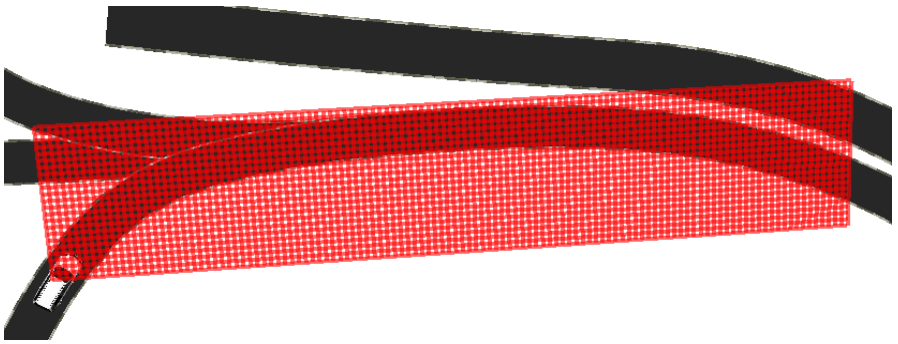


(o) A *LATTICE-SEC-TREE* results from the procedure of *focused construction and sampling*

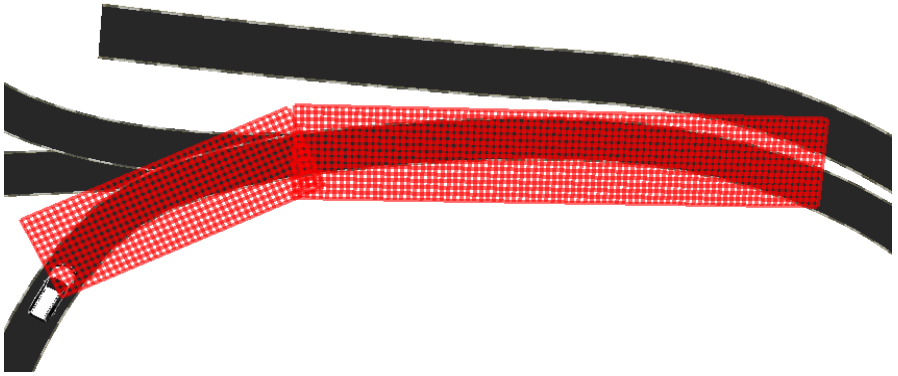


(p) After the operation of *pruning*, the *LatticeSecVec* is obtained.

Figure 3.3: An example of the construction process of the spatial horizon.



(a) One grid sampled on the road segment within a given travel distance.



(b) Two grids sampled on the road segment within a given travel distance.

Figure 3.4: Uniform sampling on a curved lane in $X - Y$ space.

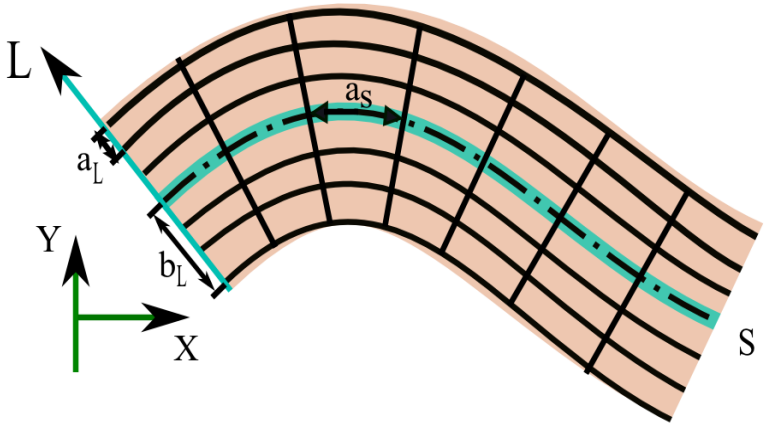



Figure 3.5: The $SL$ coordinate system laid over the road segment in $X - Y$ space.

where

$$\begin{aligned} x_n(s,l) &= x_n(s,0) + lcos(\theta_n(s,0) + \tfrac{\pi}{2}) \\ y_n(s,l) &= y_n(s,0) + lsin(\theta_n(s,0) + \tfrac{\pi}{2}) \\ \theta_n(s,l) &= \theta_n(s,0) \\ \kappa_n(s,l) &= (\kappa_n(s,0)^{-1} - l)^{-1}. \end{aligned} \tag{3.2}$$

Note that $(s,0)$ represents the coordinates of the points on the center of the lane. Given the expression of the center in terms of $(x,y,\theta,\kappa)$, the four elements of the spatial node can be easily calculated according to Equation 3.2.

The representation of the spatial node indicates that the heading $\theta$ of the vehicle at the sampled position is constrained to be parallel to the center line. This assumption is valid from the perspective that it mimics the common behaviour of human drivers in highway driving scenarios. Moreover, it results in a lattice of a small size and thus makes the planning more computationally tractable. To further decrease the computational complexity, the curvature $\kappa$ of the path at the sampled position is also assumed to be unique and is determined based on the center line. The curvature is the rate of change of heading $\theta$ with respect to change in arc length of the path and is equal to the reciprocal of the turning radius of the vehicle. Sampling the $SL$ space using a regular grid results in a set of spatial nodes denoted as $\{\hat{n}_{i_s j_l}\}$ with each $\hat{n}_{i_s j_l}$ representing a point $(s_i, l_j)$ in the $SL$ frame. Given the longitudinal and lateral intervals, i.e., $a_S$ and $a_L$ (cf. Figure 3.5), $s_i$ and $l_j$ are defined using the affine functions:

$$\begin{aligned} s_i &= a_s i \\ l_j &= a_l j - b_l. \end{aligned} \tag{3.3}$$

The warped grid in Figure 3.5 gives an instance of such a mapping.

In addition, the current posture (the collection of $x, y, \theta$ and $\kappa$) of the vehicle should also be defined as a spatial node which is denoted as $\hat{n}_{ego}$.

### 3.2.3 Path Model

As a basis for the path model design, the dynamic model of the vehicle is given as:

$$\begin{aligned} \dot{x}(t) &= V(t)cos\theta(t) \\ \dot{y}(t) &= V(t)sin\theta(t) \\ \dot{\theta}(t) &= V(t)\kappa \\ \dot{V}(t) &= a(t) \\ \kappa(t) &= u_1(t) \\ \alpha(t) &= u_2(t). \end{aligned} \tag{3.4}$$

The vehicle states include position $(x,y)$, heading $\theta$, curvature $\kappa$, speed $V$ and acceleration $\alpha$. Desired accelerations $u_2$ and curvatures $u_1$ are the control inputs to the vehicle system.

As the path model and the speed trajectory are generated separately, it is necessary to ignore the speed-related states of the vehicle model temporarily and apply a change of variable from time $t$ to distance $s$. Let the initial distance be set to zero. Integrating the re-parametrised equations renders:

$$\begin{aligned} x(s) &= \int_0^s cos\theta(s)ds \\ y(s) &= \int_0^s sin\theta(s)ds \\ \theta(s) &= \int_0^s \kappa(s)ds \\ \kappa(s) &= u(s). \end{aligned} \tag{3.5}$$

As can be seen from Equation 3.5, $x$, $y$ and $\theta$ depend directly or indirectly on $\kappa$. Consequently, the problem of path model design is equivalent to that of curvature selection. The latter has already been discussed a lot in the literature (cf. Section 2.1.3.2). In this work, the curvature cubic polynomial versus arc length is adopted to ensure steering continuity. The application of higher-degree polynomials for achieving the continuity in higher-order derivatives of the curvature is left for future work. It is noteworthy that a curve with its curvature being a polynomial of its arc length is called a polynomial spiral in [77]. Therefore, the path traced out by the curvature cubic polynomial is a cubic polynomial spiral. Let the coefficient vector of the curvature polynomial be denoted as $\vec{p}$. The polynomial spiral path is represented as:

$$\vec{\gamma}_c(s) = (x_c(\vec{p}, s), y_c(\vec{p}, s), \theta_c(\vec{p}, s), \kappa_c(\vec{p}, s))^T, s \in [0, s_T] \tag{3.6}$$

where

$$\begin{aligned} x_c(\vec{p}, s) &= \int_0^{s_T} \cos\theta(s)ds = x_0 + \int_0^{s_T} \cos(p_0 s + p_1 \tfrac{s^2}{2} + p_2 \tfrac{s^3}{3} + a_3 \tfrac{s^4}{4})ds \\ y_c(\vec{p}, s) &= \int_0^{s_T} \sin\theta(s)ds = y_0 + \int_0^{s_T} \sin(p_0 s + p_1 \tfrac{s^2}{2} + p_2 \tfrac{s^3}{3} + p_3 \tfrac{s^4}{4})ds \\ \theta_c(\vec{p}, s) &= p_0 s + p_1 \tfrac{s^2}{2} + p_2 \tfrac{s^3}{3} + p_3 \tfrac{s^4}{4} \\ \kappa_c(\vec{p}, s) &= p_0 + p_1 s + p_2 s^2 + p_3 s^3. \end{aligned} \tag{3.7}$$

Augmenting $\vec{p}$ with another unknown $s_T$ brings about an adjoined parameter vector:

$$\vec{q} = (\vec{p}, s_T). \tag{3.8}$$

Given a path primitive with its endpoints being a pair of spatial nodes $(\hat{n}_0, \hat{n}_1)$, its boundary states are summarised in a vector-valued function:

$$\vec{G}(\vec{q}) = \begin{cases} x_c(\vec{p}, s_T) - x_1 + x_0 \\ y_c(\vec{p}, s_T) - y_1 + y_0 \\ \theta_c(\vec{p}, s_T) - \theta_1 + \theta_0 \\ \kappa_c(\vec{p}, s_T) - \kappa_1 \\ \kappa_c(\vec{p}, 0) - \kappa_0. \end{cases} \tag{3.9}$$

Thus, the boundary constraints are given as:

$$\vec{0} = \vec{G}(\vec{\hat{q}}) = \vec{g}_{goal}. \tag{3.10}$$

Consequently, the path representation problem can be interpreted as finding a vector value $\vec{\hat{q}}$, that zeros $\vec{G}$. As the amount of boundary condition equations is equivalent to that of the unknowns, the problem is solvable and its solution is unique. Due to the non-linear nature of the problem, it can only be solved numerically. Therefore, a gradient-descent search is applied in this work to find an approximate solution curve. Given an initial guess of $\vec{q}$, i.e., $\vec{q}_{init}$, the main procedure of the gradient-descent search is:

$$\begin{aligned} \vec{g} &\leftarrow \vec{G}(\vec{q}_{init}) \\ \vec{J} &\leftarrow \vec{J}_{\vec{G}}(\vec{q}_{init}) \\ \triangle\vec{g} &\leftarrow \vec{g}_{goal} - \vec{g} \\ \triangle\vec{q} &\leftarrow \vec{J}^{-1} \triangle \vec{g} \\ \vec{q}_{init} &\leftarrow \vec{q}_{init} + \triangle\vec{q} \end{aligned} \tag{3.11}$$

where $\vec{J}_{\vec{G}}(\vec{q}_{init})$ refers to the Jacobian matrix of $\vec{G}$ at the point of $\vec{q}_{init}$. In this way, $\vec{q}_{init}$ is iteratively updated by a better estimation of $\vec{\hat{q}}$. Such procedure is repeated until $\triangle\vec{g}$ becomes sufficiently small for the path construction purpose, or, when a maximum number of iterations is reached, implying a divergence of the algorithm due to an initial guess far from $\vec{\hat{q}}$.

In practical implementations, however, several issues concerning convergence, efficiency and accuracy of the algorithm arise. There is a detailed coverage of those issues and the techniques designed to deal with them in [46] [78][23]. All those techniques are adopted in this work. In the following, their general ideas are presented. For more details, please refer to [46][78][23].

### INITIAL GUESS TABLE

An initial guess $\vec{q_{init}}$ that is near to the exact solution $\vec{\hat{q}}$ is essential for a fast convergence of the gradient descent search. To that end, the initial guess table is precomputed in

which each entry stores the curve solution for one specific boundary condition. The initial guesses of the records themselves are generated using the relaxation method (cf. [46]). The gradient-descent algorithm is then applied to refine the rough guess so that it may converge with $\vec{\hat{q}}$. In runtime, the initial guess for a path spiral with arbitrary boundary constraints can be obtained by using a simple nearest-neighbour lookup into the guess table.

### NUMERICAL APPROXIMATION OF FRESNEL INTEGRALS

The expressions of the position coordinates, i.e., $x(s)$ and $y(s)$, take the form of generalized Fresnel integrals (cf. Equation 3.7 ) for which there are no closed-form representations. Consequently, it is hard to evaluate them and to compute their gradients with respect to the parameter vector. As a result, Simpson's rule and the trapezoidal rule are adopted to generate numerical approximations of the original transcendental integrals (cf. [46][23]). Simpson's rule is given as:

$$\int_a^b f(x)dx \approx \frac{b-a}{6}[f(a) + 4f(\frac{a+b}{2}) + f(b)]. \tag{3.12}$$

The composite Simpson's rule is the following approximation:

$$\int_a^b f(x)dx \approx \frac{h}{3}[f(x_0) + 2\sum_{j=1}^{n/2-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)] \tag{3.13}$$

where $x_j = a + jh$ for $j = 0, 1, ..., n-1, n$ with $h = (b-a)/n$. By this formula, it is supposed that the interval $[a, b]$ is broken up into $n/2$ subintervals, with $n$ an even number. The composite Simpson's rule is applied for each subinterval. This expression is used to provide an approximate target position of the path for the gradient descent search, while its symbolic differentiation is adopted for computing the Jacobian matrix.

The trapezoidal rule is given as:

$$\int_a^b f(x)dx \approx \frac{b-a}{2}[f(a) + f(b)]. \tag{3.14}$$

The trapezoidal rule can be implemented in two kinds of composite formulations. One integrates over grid points resulting from uniform sampling in the domain $[a, b]$, while the other non-uniform. Given $n+1$ points in the interval $[a, b]$ with the equal spacing $h = (b-a)/n$, the former is expressed as:

$$\int_a^b f(x)dx \approx \frac{h}{2}[f(x_0) + 2\sum_{j=1}^{n-1} f(x_j) + f(x_n)] \tag{3.15}$$

where $x_j = a + jh$ for $j = 0, 1, ..., n-1, n$. As for the latter, it follows that

$$\int_a^b f(x)dx \approx \frac{1}{2}\sum_{j=0}^{n+1}(x_{j+1}-x_j)[f(x_{j+1})+f(x_j)] \tag{3.16}$$

where $a = x_0 < x_1 ... < x_{n-1} < x_n = b$. Note that for successive points, one can have the sampling formula

$$\int_a^{x_{j+1}} f(x)dx \approx \int_a^{x_j} f(x)dx + \frac{1}{2}(x_{j+1}-x_j)[f(x_{j+1})+f(x_j)]. \tag{3.17}$$

which applies in all the composite formulations based on the trapezoidal rule and Simpson's rule. Following [46][78][23], the proposed planner applies the sampling formula based on Equation 3.15 for path sampling. Unlike in [46][78][23] where the path samples can also be used for the evaluation of the trajectory cost, the trajectory construction strategy employed in the proposed planner makes it inconvenient to reuse the path sampling result. Consequently, Equation 3.15 is adopted for trajectory sampling in this work. More explanations can be found in Chapter 4. Simpson's rule is based on quadratic interpolation, while the trapezoidal method follows linear interpolation. Accordingly, the former has faster convergence than the latter for most functions which are twice continuously differentiable as in the case of the polynomial path employed in the proposed planner. As a result, the composite Simpson's rule with a smaller $n$ can have the same level of accuracy as the composite trapezoidal integration with a larger one. This advantage of Simpson's rule accounts for its application in the path optimization. In the case of path and trajectory sampling, the subinterval of the composite formulation is so small that Simpson's rule and the trapezoidal method are almost equal in yielding satisfactory results for each subinterval. As the trapezoidal rule only requires calculating two points for one subinterval, rather than three points as in the case of Simpson's rule, it is more efficient for sampling purpose. Since the integrands $\cos\theta(s)$ and $\sin\theta(s)$ (cf. Equation 3.7) are smooth and not oscillatory for most road paths, the numerical integrations like Simpson's rule and the trapezoidal rule have sufficient accuracy in most cases for their application in solving and evaluating path spirals.

### REFORMULATION OF CURVATURE CUBIC POLYNOMIALS

The formulation of the curvature polynomial given in Equation 3.7 may introduce round-off errors due to potentially large discrepancies between the coefficients, especially in the case of $p_1$ and $p_3$ [23]. That issue becomes problematic when it comes to the calculation of the inverse of the Jacobian matrix during the gradient descent search (cf. Equation 3.11). To that end, a reformulation of the curvature polynomial is given as:

$$\kappa_{c_a}(\vec{a}, s) = p_0(\vec{a}) + p_1(\vec{a})s + p_2(\vec{a})s^2 + p_3(\vec{a})s^3. \tag{3.18}$$

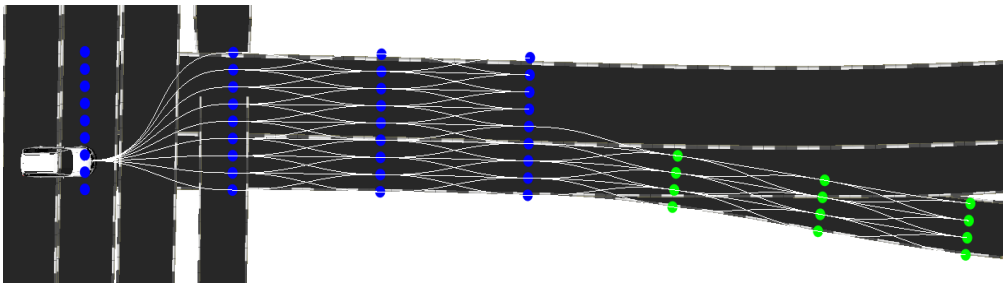


Figure 3.6: An example of the spatial graph. For path edges within the state lattice: $\{(m,n)\} = \{(1,-1),(1,0),(1,1)\}$. $\hat{n}_{ego}$ is connected to one layer of nodes.

where

$$\begin{aligned} \vec{a} &= (a_0, a_1, a_2, a_3)^T \\ a_0 &= \kappa_c(\vec{p}, 0) \\ a_1 &= \kappa_c(\vec{p}, \tfrac{s_T}{3}) \\ a_2 &= \kappa_c(\vec{p}, \tfrac{2s_T}{3}) \\ a_3 &= \kappa_c(\vec{p}, s_T). \end{aligned} \tag{3.19}$$

Such reformulation is implemented in [78] and followed by [23] to force the coefficients to be of comparable scale.

### 3.2.4 Connectivity Pattern

Given the spatial nodes and the path model, it is time to introduce the *connectivity pattern* that specifies whether a path edge should be constructed to connect two arbitrary spatial nodes. To that end, a set of integer pairs $\{(m,n)\}$ is defined where $m = 1, ..., M$, and $n = -(N-1)/2, -(N-1)/2+1, ..., (N-1)/2$. In this way, a source node $\hat{n}_{ij}$ can be theoretically connected to $M \times N$ target nodes given as $\hat{n}_{(i+m)(j+n)}$, where $(m,n) \in \{(m,n)\}$. It is noteworthy that whether a path edge can be constructed in practice is still subject to the existence of the related target node.

Besides, another connectivity pattern is necessary for defining what spatial nodes the starting node of the vehicle $\hat{n}_{ego}$ should be connected to.

So far, the spatial sampling is finished. Figure 3.6 demonstrates an example of the spatial graph.

## 3.3 Temporal Sampling

This section introduces the temporal space to augment the spatial graph constructed in last section such that a spatiotemporal state lattice can be obtained. Here the temporal space may optionally include the dimensions of time, speed, acceleration and jerk. Recall that a change of variable from time $t$ to distance $s$ is applied for the path representation

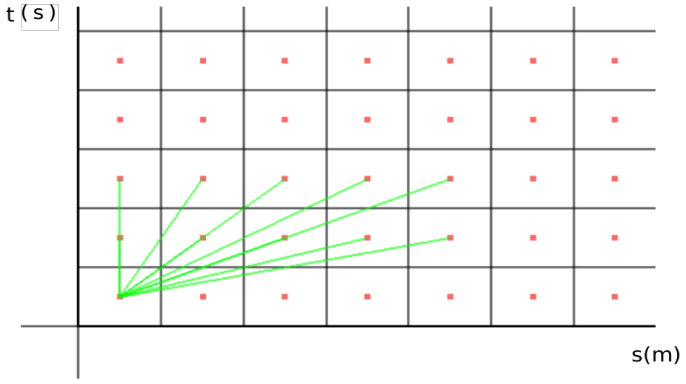



Figure 3.7: A simplified formulation of the spatiotemporal sampling. $s$ denotes the arc length of the path curve. $t$ indicates time. Red points are the representative points of the grid cells. Green line segments demonstrate the trajectory edges outgoing from the node at the left-bottom corner.

and separates the temporal space and the spatial one (cf. Section 3.2.3). Correspondingly, the essential task of temporal sampling is to find one or several expressions of $s$ in terms of $t$ for each path edge in the spatial graph.

In order to guarantee a certain level of trajectory smoothness, the acceleration cubic polynomial is employed in this work. In addition, the jerk continuity at the joints of successive trajectory primitives is ensured (assuming that the jerk of starting state of the vehicle is given). As a result, the speed trajectory can be given as:

$$\vec{\gamma}_t(s) = (s_t(\vec{a}, s), v_t(\vec{a}, s), \alpha_t(\vec{a}, s), jerk_t(\vec{a}, s))^T, t \in [0, t_T] \tag{3.20}$$

where

$$\begin{aligned} s_t(\vec{a}, t) &= \int_0^t v_t dt = v_0 t + a_0 \frac{t^2}{2} + a_1 \frac{t^3}{6} + a_2 \frac{t^4}{12} + a_3 \frac{t^5}{20} \\ v_t(\vec{a}, t) &= \int_0^t \alpha_t dt = v_0 + a_0 t + a_1 \frac{t^2}{2} + a_2 \frac{t^3}{3} + a_3 \frac{t^4}{4} \\ \alpha_t(\vec{a}, t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ jerk_t(\vec{a}, t) &= a_1 + 2a_2 t + 3a_3 t^2. \end{aligned} \tag{3.21}$$

In the equations above, $\vec{a} = (a_0, a_1, a_2, a_3)$, and $v, j$ and $t_T$ denote speed, jerk and the trajectory duration respectively.

Given the speed trajectory model, it is time to design the sampling pattern of the temporal space to enable trajectory construction. Let a simplified formulation of the problem be considered first where the spatial space is reduced to one dimension $s$. Here $s$ refers to the arc length of the path curve. Without loss of generality, the path is discretized into a sequence of evenly-spaced points, as is shown in Figure 3.7. A line segment joining two points can be imagined as a path edge in the original spatial space.

To augment the $s$ dimension with the temporal space, one intuitive idea is to sample the time dimension using a regular pattern (cf. Figure 3.7) and to effect inverse trajectory generation based on the states of given endpoints. In this setting, the endpoint constraints are:

$$\begin{aligned} s_t(\vec{a}, t_T) &= s_T \\ \alpha_t(\vec{a}, 0) &= \alpha_0 \\ jerk_t(\vec{a}, 0) &= jerk_0. \end{aligned} \tag{3.22}$$

It is noteworthy that $s_T$, $\alpha_0$, $jerk_0$ and $v_0$ are given by the distance interval and the source endpoint. To solve for the four coefficients of the acceleration polynomial, i.e., $\vec{a}$, four independent equations are required. However, there are only three of them in Equation 3.22. Consequently, one more endpoint constraint needs to be established. Accordingly, one more dimension, either speed or acceleration or jerk, should be introduced into the temporal space. Should this dimension also be sampled using a regular pattern, a 2D grid of the temporal space is obtained.

In practice, however, this 2D uniform grid is problematic. On one hand, it is highly likely that the trajectory primitive generated in this way is dynamically infeasible. As a result, it is necessary for the uniform sampling to be dense enough to guarantee a certain level of completeness. On the other hand, specific speeds and accelerations are preferable in some traffic scenarios. For example, velocity keeping might require that the vehicle speed approaches the speed limit of the current road. Another instance occurs in vehicle following, where the acceleration of the ego-vehicle has to be adjusted according to its distance and speed relative to the followed vehicle. From this perspective, dense sampling is also necessary so that a grid point would coincide with, or be near to, the demanded state. However, dense sampling would inevitably entail prohibitive computational cost. To mitigate that issue, a forward generation approach is adopted in this work. It works like this:

- A set of acceleration profiles are selected according to the vehicle physics and scenario-dependant requirements. It may contain profiles concerning e.g., maximum and minimum accelerations and the speed limit of the current road. In this way, a certain level of trajectory diversity can be guaranteed with a computationally tractable graph.

- All the acceleration profiles are associated with each of the path edges, which results in trajectory edges with different target states incoming at the same spatial node.

- To further decrease the computational cost, those resultant trajectory edges are subject to pruning at the spatial node via a uniform grid in terms of speed and

time. That is, only one trajectory edge is allowed to be further expanded if there are several of them incoming at the same time-speed grid cell. It should be pointed out that such measure might deteriorate the trajectory diversity to some extent. To alleviate this situation, trajectory edges with different acceleration profiles are defined not to compete against each other. This strategy is equivalent to regarding the acceleration profile as one dimension and incorporating it into the temporal space for sampling.

As a result of the temporal sampling approach presented above, a node in the resultant state lattice is denoted as $n(i_s, i_l, i_\alpha, i_v, i_t)$. In this denotation, $(i_s, i_l)$ is the index of the underlying spatial node $\hat{n}_{i_s i_l}$, $i_\alpha$ is the index of the acceleration profile applied on the trajectory primitive ending at this node, and $i_v$ and $i_t$ can be used to access the corresponding time-speed grid cell. One point worth mentioning is that the representative point of this grid cell is generated on the fly during the graph construction. In other words, the speed and time $(v, t)$ representing the time-speed grid cell remain unknown (but confined within the time and speed intervals dominated by this cell) until an operation of pruning takes place at this cell, where the speed and time of the target state of the trajectory edge that survives the pruning are assigned to $(v, t)$. That property distinguishes $v$ and $t$ from the spatial node and the acceleration profile whose values can be directly obtained base on their indexes, i.e., $i_s, i_l, i_\alpha$. Furthermore, an edge in the state lattice that joins two such nodes is also called a trajectory edge (in comparison to the path edge) and is denoted as $e_{n_k \to n_{k+1}}$.

As for the concrete representation of the acceleration profiles and their implementations in the proposed planner, more details can be found in Chapter 4.

## 3.4 Graph Search

As is mentioned previously, the proposed motion planner selects the best constraint-abiding trajectory($\tau_{opt}$) among numerous candidates ($\tau_c$). The evaluation criteria take into account the traversal cost ($G_\tau$) of the trajectory and the desirability ($H_\tau$) of its ending node ($n_\tau$) from the perspective of the next mission goal. Consequently, the selection can be mathematically formulated as:

$$\tau_{opt} = \underset{\tau \in \tau_c}{\operatorname{argmin}} (G_\tau(\tau) + H_\tau(n_\tau)). \tag{3.23}$$

It is noteworthy that the constraints are implicitly covered in Equation 3.23. The reason is twofold. On one hand, it is defined via traversal cost functions that $G_\tau(\tau) = \infty$ if $\tau$ violates the collision and physics-related constraints. On the other hand, a trajectory with a cost of $\infty$ is regarded as untraversable or infeasible and thus must be abandoned.

Note that a trajectory may consist of several trajectory edges. A trajectory consisting of $N$ trajectory edges is thus represented as:

$$\tau_N = \{e_{n_k \to n_{k+1}}\}, k = 0, 1, ..., N-1 \tag{3.24}$$

As is discussed in the previous section, a group of trajectory edges $\{e_{in}\}$ might all converge at one to-be-constructed node $n(i_s, i_l, i_\alpha, i_v, i_t)$. Such phenomenon occurs due to the application of the connectivity pattern and the nature of the forward temporal sampling approach adopted in the proposed planner. Since the representative value of one node is defined to be unique, an operation of pruning is necessary for the selection of the trajectory edge whose target endpoint should represent the node. The trajectory edge $e_{opt}$ that survives the pruning satisfies:

$$e_{opt} = \underset{e_{n_k \to n_{k+1}} \in \{e_{in}\}}{\operatorname{argmin}} (G_E(e_{n_k \to n_{k+1}}) + H_E(n_{k+1})) \tag{3.25}$$

where

$$G_E(e_{n_k \to n_{k+1}}) = \sum_{i=0}^{k} c_g(e_{n_i \to n_{i+1}}). \tag{3.26}$$

In Equation 3.25, $H_E(n_{k+1})$ evaluates the desirability of the intermediate node $n_{k+1}$ for the purpose of reaching the next mission goal. Only the criteria with regard to the time and speed of $n_{k+1}$ are taken into account in $H_E(n_{k+1})$ as other factors such as the station and latitude are the same among all the candidate nodes for one time-speed grid cell. In comparison, $H_\tau$ needs to calculate all the criteria that can be used to differentiate the target nodes which might have different stations and latitudes. There are more details about $H_\tau$ in Subsection 5.2.3. In Equation 3.26, $c_g(e_{n_i \to n_{i+1}})$ calculates the traversal cost of the edge $e_{n_i \to n_{i+1}}$. Note that here $n_{k+1}$ is the endpoint of the to-be-evaluated trajectory edge which might not end up representing the to-be-constructed node.

The final selection and the intermediate pruning operations are associated by:

$$\begin{aligned} G_\tau(\tau_N) &= G_E(e_{n_{N-1} \to n_N}) \\ H_\tau(n_{k+1}) &= H_E(n_{k+1}) + \text{COST-SPECIFIC-FOR-TARGET-NODE}. \end{aligned} \tag{3.27}$$

As can be seen from the evaluation strategy illustrated above, the search in the graph (in the sense of the evaluation and selection of the trajectory edges) is in effect carried out in the course of the graph construction process (in terms of the node construction based on the result of the evaluation). A decision should be made whenever a node is met. Besides, several nodes might be in the way of the forward construction of one

trajectory. As a result, the search for the optimal trajectory can be formulated as a dynamic programming problem. Such problems can be solved efficiently by observing the principles of optimality proposed in [79]. Concretely, the principles of optimality can be paraphrased in this context as follows:

**Principle 3.6.** *All the potential incoming edges at one node should be evaluated before its outgoing edge can be generated.*

In this way, it can be ensured that the evaluated edges are those that are believed to lead to the optimal trajectory based on the selection result at their source nodes (although they might not be able to survive the pruning at their target nodes). In other words, it avoids constructing and evaluating edges whose doomed fate can be foreseen as early as the time when their starting nodes fail in the pruning. Correspondingly, the graph search can be conducted in a more efficient fashion.

Based on the assumption that the vehicle does not move reversely, a practical principle for the graph construction is given as:

**Principle 3.7.** *The trajectory edges starting from a single station should be constructed after all the trajectory edges ending at it are constructed and evaluated.*

It is noteworthy that complying with Principle 3.7 is a sufficient albeit not necessary condition for a strategy of graph construction to observe Principle 3.6. The proposed motion planner conducts the graph construction and search in the guidance of Principle 3.7. Figure 3.8 shows an example of the set of candidate trajectories constructed by the proposed motion planner.

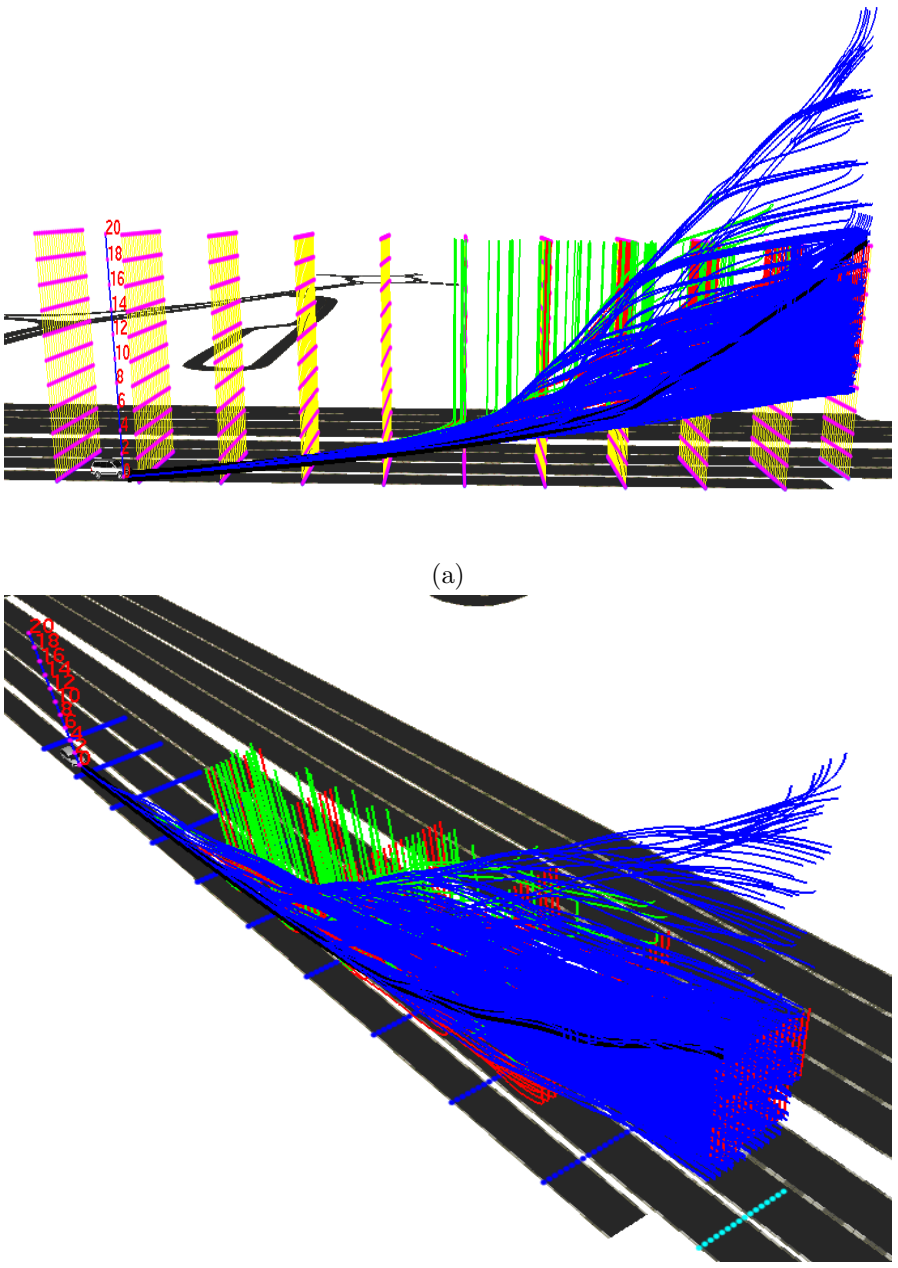
Note that the trajectory edges that start from the same station can be constructed and evaluated without following any particular order. Accordingly, parallel algorithms can be implemented to accelerate this process. More details can be found in Chapter 4 and Chapter 5.

## 3.5 Trajectory Evaluation

For the evaluation of its traversal cost, the trajectory edge $e$ is discretized into a set of samples $\{\vec{x}_i\}_{i=0}^{N}$ where $\vec{x} = (x, y, \theta, \kappa, v, \alpha, t)^T$. A set of cost functions $\{C_j\}_{j=0}^{M}$ are devised and each function is evaluated over $\{\vec{x}_i\}_{i=0}^{N}$. Accordingly, the traversal cost $c_g(e)$ can be calculated as:

$$c_g(e) = \sum_{j=0}^{M} \sum_{i=0}^{N} C_j(\vec{x}_i). \tag{3.28}$$

A list of all cost criteria applied in this work can be found in Chapter 5. In this section, only the cost functions implemented as cost maps are addressed.




(a)

(b)

Figure 3.8: An example of the set of candidate trajectories demonstrated in spatiotemporal space. The blue trajectories are those that manage to arrive at the end of the spatial horizon. The green trajectories come to a stop halfway along arbitrary edges; the ending nodes of these trajectories are called *EXTRA-SAMPLE* in this thesis (cf. Subsection 4.2.2). The red trajectories terminate at arbitrary spatiotemporal nodes that are not at the end of the planning horizon. The green and red trajectories are forced to be extended to the nominal upper time boundary of the state lattice (cf. Subsection 5.2.3). The yellow vertical line segments standing at the spatial samples represent the time axes. The magenta points indicate the time coordinates (seconds) in accordance with the numbers shown along the blue vertical line segment.

### 3.5.1 Cost Maps

Recall that a cost map is a sampled version of the cost function. As is discussed in Subsection 2.1.2, cost maps are usually employed for efficient evaluation of the trajectories. This is especially true for applications of search-based motion planning as proposed in this thesis, where tens of thousands of trajectories are evaluated. Three cost functions are implemented as cost maps in this work:

- Static obstacles: $c^{obst}_{static} = C^{obst}_{static}(x, y)$. This cost function is employed to keep the vehicle away from the static obstacles. Similar to [23], both fatal and high-cost areas are defined for each static obstacle. The fatal area is untraversable for the vehicle. It contains the area occupied by the obstacle and a dilation around the vehicle as a compensation for the vehicle heading direction (which will be further discussed later). The high-cost area defines a safe clearance between the obstacle and the vehicle to compensate for potential sensing errors. The vehicle is allowed to enter the high-cost area, albeit subject to severe punishments. Other areas in the spatial horizon are free space with a cost of zero. Figure 3.9(f) gives an instance of the cost map of static obstacles.

- Dynamic obstacles: $c^{obst}_{dynamic} = C^{obst}_{dynamic}(x, y, t)$. This function penalizes the vehicle for getting too close to the dynamic obstacles. It is evaluated based on a prediction of the future state of the dynamic obstacles. Given a specific time, the function is defined in a similar way to $C^{obst}_{static}$ albeit with some variations. Similar to [23], the amount of the dilation around dynamic obstacles depends on their distance to the ego-vehicle, their velocity and how far away they are in terms of time. This is unlike that of static obstacles which is only affected by their distance to the ego-vehicle. Such variation is applied to take into account the potential errors introduced by the sensing system as well as the obstacle detection and tracking modules. Besides, in addition to the fatal and high-cost areas as with static obstacles, the *following* area is also specified right behind the dilated dynamic obstacle to facilitate the vehicle following behaviour. The cost map of dynamic obstacles consists of several frames indexed in time. With the dynamic obstacles frozen at its timestamp, each frame is rendered like a cost map of static obstacles. Figure 3.10 displays an example of the cost map of dynamic obstacles.

- Lane centering: $c^{lane}_{static} = C^{lane}_{static}(x, y)$. By punishing the deviation of the trajectory with respect to the center of the lane, the cost function of lane centering helps to keep the vehicle stay at the lane center. Besides, it also defines a safe clearance between the road curbs and the vehicle. Lastly, it penalizes the trajectories on the lanes with oncoming traffic so that the vehicle will remain on lanes of its desired travel direction as long as it is possible to do so. Figure 3.11 demonstrates the

costs of a layer of points obtained via 2D linear interpolation on the lane centering cost map.

As the trajectory is represented in $(x, y)$ coordinates, all the cost maps are defined in $X - Y$ space for the convenience of evaluation. However, the cost functions need to be evaluated in the $SL$ frame. This necessity is straightforward for $C^{lane}_{static}$. In terms of $C^{obst}_{static}$ and $C^{obst}_{dynamic}$, it can be seen from the fact that the obstacles are dilated in the $SL$ frame (cf. Subsection 3.5.2). Figure 3.9 demonstrates an example of the construction of the cost map for static obstacles via dilation in the $SL$ frame. As a result, the grid points $(x, y)$ of the cost map need to be mapped to their counterparts $(s, l)$ for which the cost function is evaluated. As there are several cost maps, a lookup table ( the $XYSL$ map ) is constructed to accelerate the mapping. At the start of each planning cycle, the $(s, l)$ coordinates of a set of grid points $(x, y)$ sampled on the planning horizon are computed once and stored in the $XYSL$ map. Consequently, the mapping from $(x, y)$ to $(s, l)$ is carried out by implementing a 2D interpolation from the four nearest neighbours of the $(x, y)$ in the $XYSL$ map.

The construction and implementation of these cost maps in this work are similar to those presented in [23]. The main differences include:

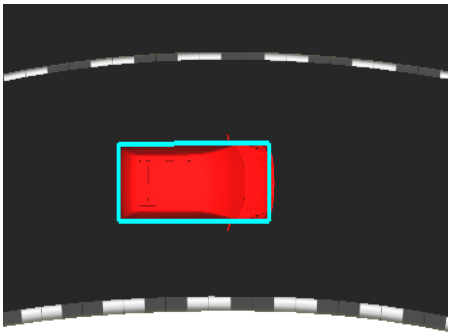
- As the planning horizon specified in this work may contain several segments, the cost map for the planning horizon has several submaps, and there is at least one submap for one segment. The same applies to $XYSMap$. In comparison, there are no submaps for the cost map constructed in [23] as the planning horizon defined there is simply one road segment with a uniform width.

- Should only one grid be constructed for a horizon segment, it would be too large in the case of e.g., a curved road as can be seen from Figure 3.4. To that end, several grids may be constructed for one segment in this work for the purposes of adapting to the road shape. This measure discounts the computational cost from the perspective of cost map construction. However, it may lead to a more expensive lookup during trajectory evaluation as each point for evaluation should first find out which submap covers it. Therefore, the amount of the submaps should be adjusted according to the practical performance in terms of accuracy and computational efficiency.

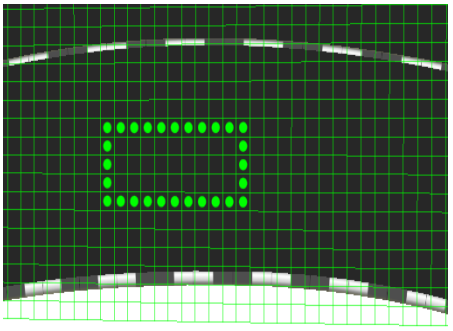
As each submap is treated in a similar way to the cost map illustrated in [23], no more accounting is repeated here. For more details, please refer to [23].

### 3.5.2 Obstacle Dilation

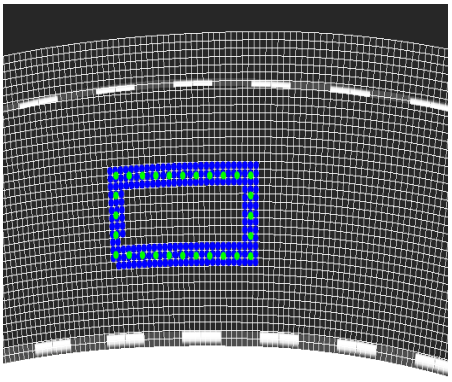
As is discussed in Subsection 2.1.2, in order to achieve a safe, flexible and efficient planning, it is necessary to dilate the obstacles according to the dimension and heading



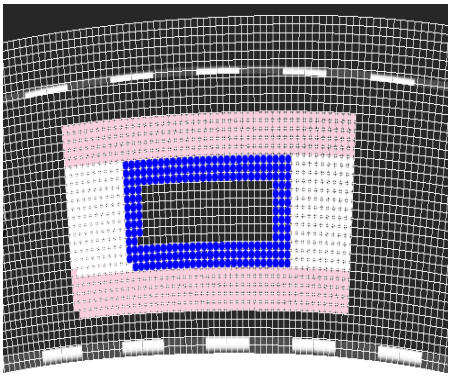
(a) A static obstacle on the road. The solid box is the bounding box of the obstacle generated by the perception system.



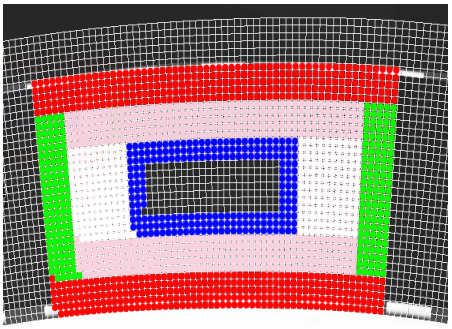
(b) The bounding box is mapped onto the $XY$ grid over which the cost map for static obstacles is defined. In the proposed planner, only the contour of the obstacle is mapped to the $SL$ grid for dilation.



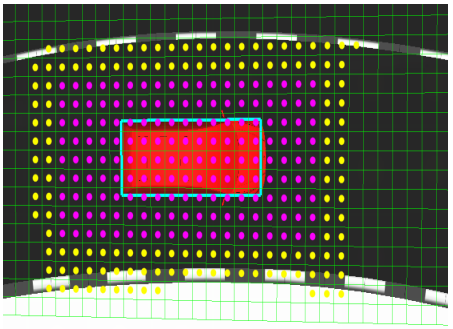
(c) The bounding box is mapped from the $XY$ grid to the $SL$ grid. In order to achieve a more accurate dilation, the discretization of the $SL$ grid is defined to be smaller than the $XY$ grid. The blue area is the mapping result.



(d) The bounding box on the $SL$ grid with the dilation of lethal area (the white and pink area). The lethal area dilation is carried out along both $S$ and $L$ directions.

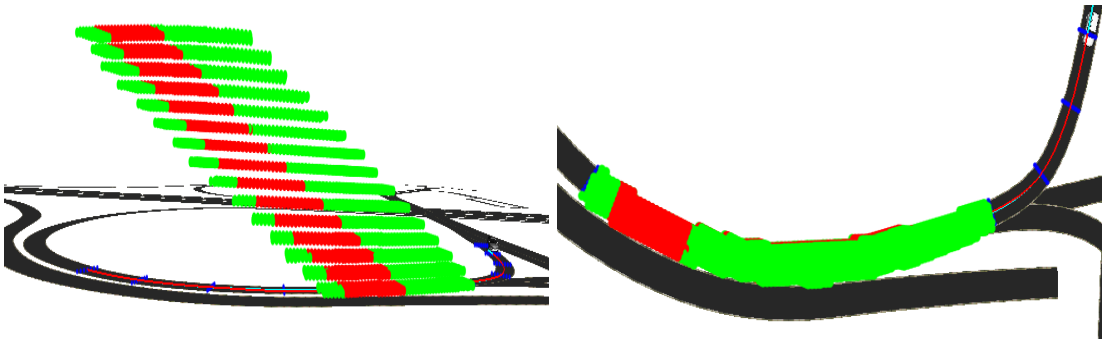


(e) The bounding box on the $SL$ grid with the dilation of lethal area and high-cost area (the green and red area). The high-cost area dilation is carried out along both $S$ and $L$ directions.



(f) The dilated bounding box on the $SL$ gird is mapped back to the $XY$ grid. This is the final result of the cost map for static obstacles.

Figure 3.9: Construction of the cost map for static obstacles.



(a) Side view of the cost map. (b) Top view of the cost map.

Figure 3.10: An example of the cost map of dynamic obstacles. The cost map contains several submaps. Only the fatal (red), high cost and following (green) areas are displayed.

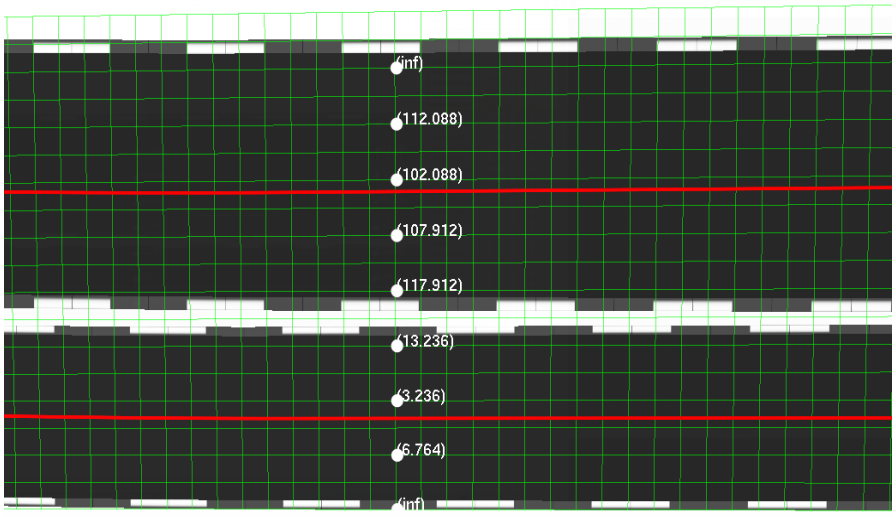



Figure 3.11: Costs of a layer of points as a result of 2D linear interpolation on the lane centering cost map. The cost map is defined over the green grid sampled on a two-lane road segment. The upper lane is the one with opposing traffic. The red lines represent the center lines of the two lanes.

of the ego-vehicle. In [23], the amount of dilation is unique and is determined based on the assumption that a maximum of six-degree deviation of the heading of the ego-vehicle from parallel to the center line (denoted as $\theta_{dev}$) can occur during highway driving. In this way, the dilation can be readily accomplished in the $SL$ coordinate system. It is also argued in [23] that in cases where the deviation is larger than the assumed six-degree, the omission of some points that are in collision can be remedied by the correct judgement for other points along the path. However, no detailed analysis is provided to justify the argument in [23].

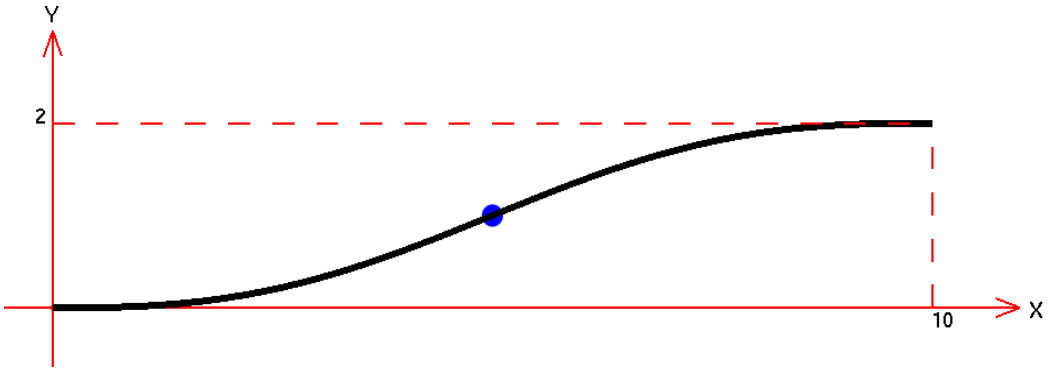
Based on this dilation strategy, an analysis is provided in the following as to whether an assumed maximum deviation (denoted as $\theta^a_{dev}$) is sufficient for the collision checking

given a specific spatial-sampling pattern.

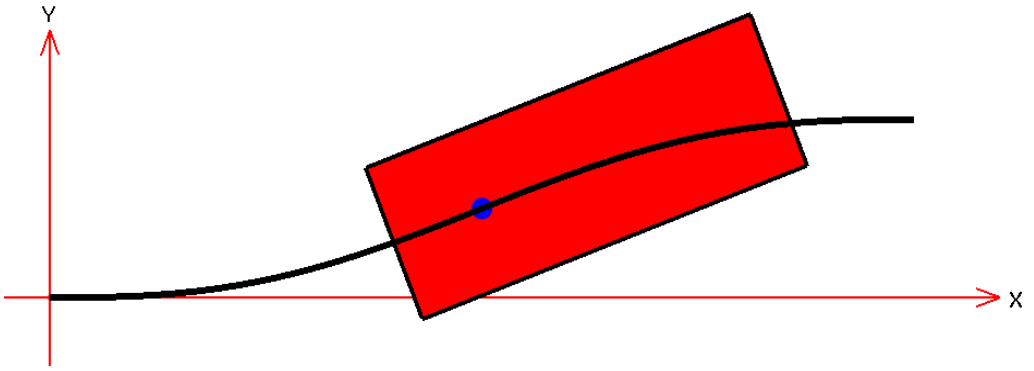
The goal of the analysis is to verify whether the dilated obstacle that is responsible for the missed collision at one point of the path can be detected to collide with another point along the same path or the paths in its neighbourhood. The path spirals may have various shapes on the driving route under a specific spatial sampling pattern. Theoretically, all those path spirals that violate the assumption about the maximum heading deviation should be subject to such verification.

As there is no closed-form solution for the calculation of the parameters of the curvature polynomial ( i.e., the path model), it is difficult to develop a general mathematical model of the analysis. Consequently, the sufficiency is analysed in a numerical way. Given a spatial sampling pattern and a uniform road segment as the planning horizon, the analysis against one path primitive in question is illustrated as follows and demonstrated in Figure 3.12.

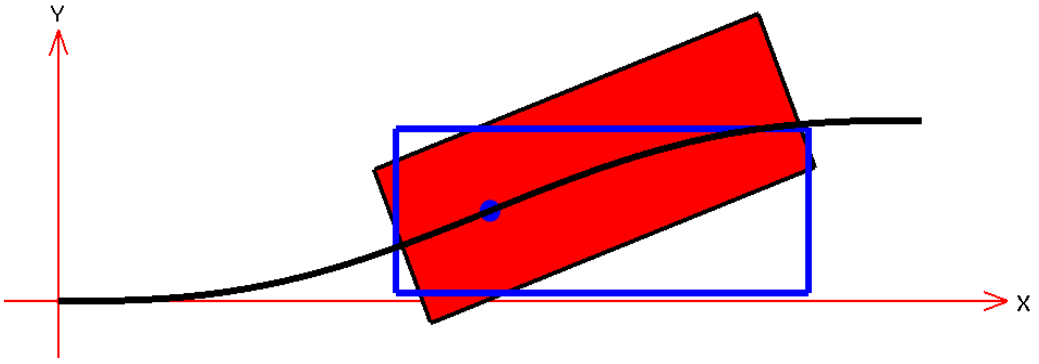
- a) A path primitive $P_p$ is given along which the maximum $\theta_{dev}$ is larger than $\theta^a_{dev}$. Other paths that are in the neighbourhood of $P_p$ are given as $P_{other}$.
- b) An arbitrary point on $P_p$ where $\theta_{dev}$ is larger than $\theta^a_{dev}$ is located and denoted as $p_{focus}$.
- c) Both the actual and dilated forms of the ego-vehicle frame are rendered at $p_{focus}$ (denoted as $C_{act}$ and $C_{dilate}$ respectively). The part of $C_{act}$ that is outside of $C_{dilate}$ is the *blind area* (denoted as $A_{blind}$) for the collision checking at $p_{focus}$. The rest of the analysis is all about verifying whether $A_{blind}$ can be covered by the collision checking at other points along $P_p$ or $P_{other}$.
- d) Should the ego-vehicle be in collision within $A_{blind}$, it would mean that there are obstacles overlapping $A_{blind}$. Given a point obstacle somewhere within $A_{blind}$, its dilated shape $obst_{dilate}$ can be rendered. Should $obst_{dilate}$ collide with $P_p$ or $P_{other}$, it can be concluded that the point obstacle can be removed from $A_{blind}$.
- e) Procedure d) is repeated until $A_{blind}$ is empty, or one point is discovered that cannot be removed from $A_{blind}$. The former indicates that $p_{focus}$ does not challenge $\theta^a_{dev}$ from the perspective of the collision checking over the entire planning horizon. The latter communicates a message that $\theta^a_{dev}$ is not sufficient for an accurate collision checking of $P_p$. Note that it is impossible to check all the points in $A_{blind}$ as the number of them is infinite. As a result, this numerical analysis can only cover a limited number of points. It should be pointed out that the configuration (i.e., heading and position) of the vehicle and the shape of the path spiral in question can be exploited further to reduce the size of $A_{blind}$.



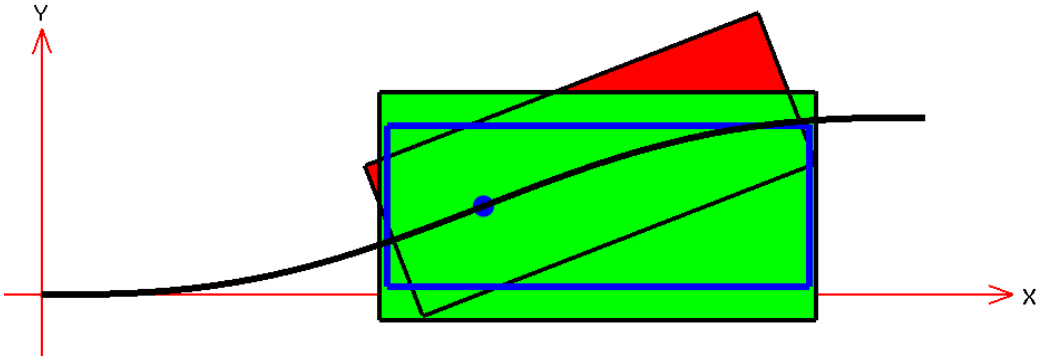

(a) A path primitive $P_p$. The blue point refers to the point $p_{focus}$ against which the analysis is conducted.




(b) The actual vehicle frame with heading tangent to the path $P_p$ at $p_{focus}$. This frame is denoted as $C_{act}$ in the analysis.

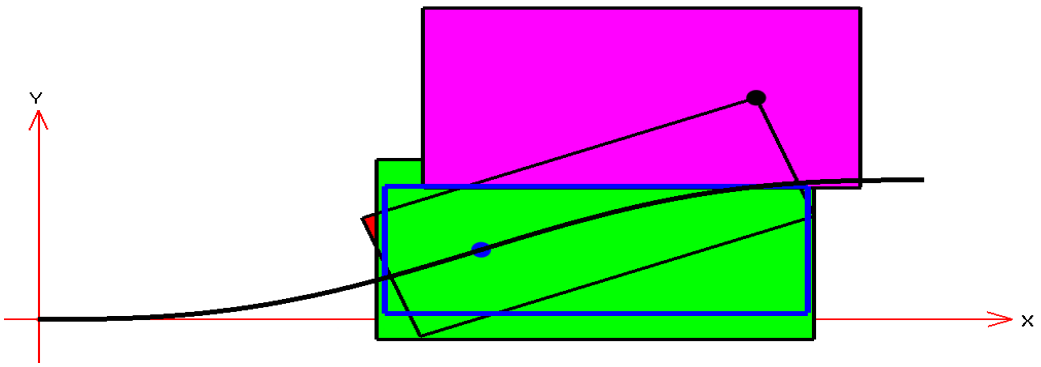



(c) The actual vehicle frame (blue box) with heading parallel to the center line of the road (here, the $x$ axis).

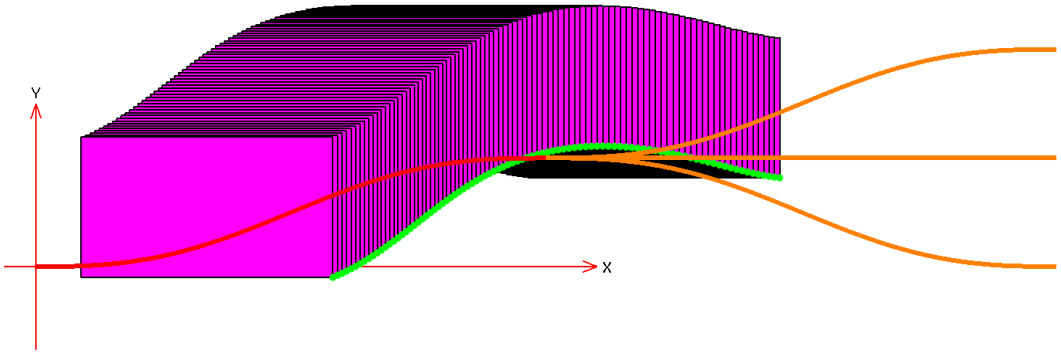



(d) The vehicle frame (green filled box) dilated according to $\theta^a_{dev}$. This frame is denoted as $C_{dilate}$ in the analysis. The red area of $C_{act}$ that still exposes outside $C_{dilate}$ is the *blind area*.

Figure 3.12: Sufficiency analysis of the dilation based on the assumed heading deviation $\theta^a_{dev}$ from parallel to the center line. $\theta^a_{dev} = 6^o$.




(e) The point obstacle dilated according to $\theta^a_{dev}$ given as a pink box. The black point at the top-right corner of $C_{act}$ refers to the point obstacle. It is denoted as $obst_{dilate}$ in the analysis. In this example, $obst_{dilate}$ is in collision with the path $P_p$, which means that the point obstacle can be detected at other points of the path. As a result, the point obstacle can be removed from the blind area.




(f) The group of $obst_{dilate}$ generated by moving $p_{focus}$ along $P_p$. The red curve $P_p$ and the orange ones $p_{other}$ are successive path primitives. The green curve is generated by connecting the bottom-right corners of the group of $obst_{dilate}$. It can be concluded in this analysis that the point obstacles that have those green points above the curves cannot be detected by any points on the demonstrated path curves

Figure 3.12: Sufficiency analysis of the dilation based on the assumed heading deviation $\theta^a_{dev}$ from parallel to the center line. $\theta^a_{dev} = 6^o$.

By applying the analysis procedures mentioned above on all the points of interest along $P_p$, the sufficiency of $\theta^a_{dev}$ for the collision checking of $P_p$ can be evaluated. It is noteworthy that the whole analysis can be automated. This automation is left for future work.

In this way, the minimum $\theta^a_{dev}$ that can ensure a sufficient collision checking (from the perspective of the numerical analysis) for a given spatial-sampling pattern can be derived. If the minimum heading deviations required by different path types have large discrepancies among each other, multiple cost maps can be constructed with each responding to one particular requirement. In comparison with the collision checking with one cost map, the application of multi cost maps can enhance the flexibility and safety of the planning, albeit at the expense of computational efficiency.

## 3.6 Summary

In this chapter, the overall planning strategy was presented. The main ideas are summarised as follows:

- Planning horizon: guidelines for specifying the planning horizon and the duration of the planning cycle were outlined. A novel approach to specify the spatial horizon including several road segments was proposed. This method can adapt to various road layouts and thus improves the planning consistency and smoothness.

- Spatiotemporal sampling: deterministic sampling is adopted in the proposed planner. The spatial space and the temporal one are sampled separately. In terms of spatial sampling, a uniform grid is constructed on a lane-adapted coordinate system. The heading and curvature of the trajectory at each grid point are defined by referring to the center line of the lane at its station. A cubic polynomial spiral is employed to join arbitrary two grid points of which the connectivity is permitted by a given connectivity pattern. As for sampling in the temporal space, a regular grid of speed and time augmented by a space of several acceleration profiles is used. The representative value for each cell of the resultant grid is determined on the fly by applying acceleration profiles on the path primitives and selecting the best endpoint among those that fall into the same speed-grid cell. Such strategy is similar in spirit to the resolution equivalent grid proposed in [80] [81].

- Graph search: principles of optimality in dealing with dynamic programming problems serve as the basic guidelines in graph construction and search. The parallel algorithm is implemented to accelerate the search, which is further discussed in the next two chapters.

- Trajectory evaluation: Three cost maps are constructed for efficient trajectory evaluation. They are cost maps for lane centering, static obstacles and dynamic obstacles. The obstacles are dilated in the lane-adapted coordinate system based on the assumption of a maximum deviation of the heading of the vehicle from parallel to the center line. The sufficiency of the collision checking scheme from the perspective of missing no potential collision is analysed in a numerical way.

According to the evaluation criteria for motion planners (cf. Section 1.3), this search-based planning method is employed here based mainly on the following considerations:

- Search-based motion planners can achieve greater extent of optimality and completeness compared to those using only terminal states.

- Although the computational complexity of the former is larger than that of the latter, the issue can be mitigated by employing parallel algorithms to accelerate the costly graph search.

- The proposed planning strategy assumes that the heading and curvature of the trajectory at one state sample are unique. Besides, the path primitive is confined to a specific functional. In this way, the search space gets smaller and thus the computational cost gets further reduced.

Besides, the application of acceleration cubic polynomials for the speed trajectory generation improves the performance of the motion planner in terms of the feasibility criterion. The next chapter addresses this issue.

# Chapter 4

# Acceleration Profiles for Smooth Trajectories

As is discussed in previous chapters, acceleration profiles are associated with the path edges to generate trajectory edges. This chapter introduces the acceleration profiles applied in the proposed planner. The main consideration in designing the acceleration profiles is that they should guarantee a certain level of smoothness of the trajectories. This requirement poses three challenges to the construction of the trajectories. Firstly, the acceleration profiles applied on the path edges should be smooth themselves. Secondly, successive trajectory edges should have a smooth transition, i.e., there are no jumps in terms of acceleration and speed at the joint of two successive edges. Lastly, the trajectories generated based on consecutive planning horizons should keep consistent to some extent. The first two problems are covered in this chapter; the last one is discussed in the next chapter. Section 4.1 recalls the derivation of the formulae for trajectories with the minimum jerk level (i.e., smooth trajectories), setting a foundation for the rest of this chapter. Section 4.2 presents the characteristics of the proposed acceleration profiles, such as the formulae of their representations, their boundary conditions and the evaluation of their feasibilities. After that, Section 4.3 illustrates how the acceleration profiles are associated with the path edges. The related principles and algorithms are also described. Finally, Section 4.4 evaluates the performance of the proposed acceleration profiles and the application strategy. The evaluation is conducted by comparing the trajectories generated by the proposed planner and those calculated based on the theory of smooth trajectories presented in Section 4.1.

## 4.1 Smooth Trajectories

It is suggested in [82] that the smoothness of a trajectory can be quantified in terms of jerk. Concretely, given a one-dimensional trajectory expressed in the form of $x(t)$ whose

domain is $[t_0, t_f]$, its smoothness can be measured by the jerk cost:

$$\int_{t_0}^{t_f} \dddot{x}(t)^2 dt \tag{4.1}$$

where

$$\dddot{x}(t) = \frac{\mathrm{d}^3\mathrm{x}(\mathrm{t})}{\mathrm{dt}^3} = jerk. \tag{4.2}$$

It is pointed out in [82] that the trajectory that most smoothly traverses from $x(t_0)$ to $x(t_f)$ is the one that has the minimum jerk cost. Accordingly, the problem of solving for a smooth trajectory can be formulated as finding $x(t)$ that satisfies given boundary conditions and at the same time, minimizes the cost functional:

$$J(x(t)) = \frac{1}{2}\int_{t_0}^{t_f} \dddot{x}(t)^2 dt. \tag{4.3}$$

The technique of calculus of variations is adopted in [82] to locate the critical point for the minimum of $J(x(t))$. Its main idea is to find $x(t)$ that satisfies:

$$\left.\frac{dJ(x(t)+\varepsilon\eta(t))}{\varepsilon}\right|_{(\varepsilon=0)} = 0 \tag{4.4}$$

where $\eta(t)$ is a function of t, and $\varepsilon$ is a real number. Adding the variation $\varepsilon\eta(t)$ to $x(t)$ causes a small perturbation to the functional $J(x(t))$ at $x(t)$ .

By associating Equation 4.3 and Equation 4.4 one can have:

$$\begin{array}{lcl} J(x+\varepsilon\eta) & = & \frac{1}{2}\int_{t_0}^{t_f}(\dddot{x}+\varepsilon\dddot{\eta})^2 dt \\ \frac{dJ(x(t)+\varepsilon\eta(t))}{\varepsilon} & = & \int_{t_0}^{t_f}(\dddot{x}+\varepsilon\dddot{\eta})\dddot{\eta}\, dt \\ \left.\frac{dJ(x(t)+\varepsilon\eta(t))}{\varepsilon}\right|_{(\varepsilon=0)} & = & \int_{t_0}^{t_f}\dddot{x}\dddot{\eta}\, dt. \end{array} \tag{4.5}$$

There is a theorem in calculus called integration by parts which is used to transform the integral of a product of functions to the integral of their derivative and antiderivative. Given two functions $u(x)$ and $v(s)$ and their differentials $du = \dot{u}(x), dv = \dot{v}(x)$, integration by parts states that :

$$\int u(x)\dot{v}(x)dx = u(x)v(x) - \int \dot{u}(x)v(x)dx. \tag{4.6}$$

The last integral in Equation 4.5 can be reformulated by repeatedly applying the technique of integration by parts:

$$\begin{array}{lcl} \int_{t_0}^{t_f}\dddot{x}\dddot{\eta}\, dt & = & \ddot{x}\dddot{\eta}|_{t_0}^{t_f} - \int_{t_0}^{t_f} x^{(4)}\ddot{\eta} dt \\ & = & \ddot{x}\dddot{\eta}|_{t_0}^{t_f} - (x^{(4)}\dot{\eta}|_{t_0}^{t_f} - \int_{t_0}^{t_f} x^{(5)}\dot{\eta} dt) \\ & = & \ddot{x}\dddot{\eta}|_{t_0}^{t_f} - x^{(4)}\dot{\eta}|_{t_0}^{t_f} + x^{(5)}\eta|_{t_0}^{t_f} - \int_{t_0}^{t_f} x^{(6)}\eta dt. \end{array} \tag{4.7}$$

$\eta(t)$ is assumed to have the following properties:

$$\begin{aligned} \eta(t_0) &= 0 \\ \dot{\eta}(t_0) &= 0 \\ \ddot{\eta}(t_0) &= 0 \\ \eta(t_f) &= 0 \\ \dot{\eta}(t_f) &= 0 \\ \ddot{\eta}(t_f) &= 0, \end{aligned} \tag{4.8}$$

By associating Equation 4.4, Equation 4.7 and Equation 4.8, one can have:

$$\int_{t_0}^{t_f} x^{(6)} \eta dt = 0. \tag{4.9}$$

As Equation 4.9 must hold true for any $\eta(t)$, it follows:

$$x^{(6)} = 0. \tag{4.10}$$

It can be seen from Equation 4.10 that the function $x(t)$ whose sixth derivative is equal to zero can render a minimum-jerk trajectory, i.e., the most smooth trajectory. Correspondingly, the general representation of the minimum-jerk trajectory is given as a position quintic polynomial which can also be regarded as the integration of the antiderivative of an acceleration cubic polynomial:

$$x(t) = p_0 + p_1 t + p_2 t^2 + p_3 t^3 + p_4 t^4 + p_5 t^5 \tag{4.11}$$

where the parameters $p_0, p_1, p_2, p_3, p_4, p_5$ can be determined by forcing $x(t)$ to satisfy the boundary conditions of the trajectory. If there are insufficient boundary constraints for calculating all the parameters, the trajectory will degenerate from the quintic polynomial to polynomials with a lower degree.

## 4.2 General Types of Acceleration Profiles Applied in the Planner

In the proposed planner, the acceleration profiles are associated with the paths to generate trajectories. This operation results in trajectories of $s(t)$. From this perspective, the trajectories can be regarded as one-dimensional trajectories with the dimension referring to the arclength of the path. Accordingly, the conclusion of the theoretical analysis illustrated in Section 4.1 can be applied to the construction of smooth trajectories in this context.

Due to the sampling nature of the state lattice, a complete trajectory generated within one planning horizon is composed of multiple pieces. Each piece is constructed

by applying one of the acceleration profiles on a path segment. This section presents the acceleration profiles devised and applied in the proposed planner. As to how they are applied on path edges and connected with each other during graph construction, it is covered in the next section.

In general, two types of polynomials are adopted for the acceleration profiles applied in the proposed planner, i.e., acceleration cubic polynomial and acceleration constant (which can also be regarded as position quadratic polynomial). The former is formulated as:

$$\begin{array}{lcl} s(t) & = & s_0 + v_0 t + \frac{1}{2}p_0 t^2 + \frac{1}{6}p_1 t^3 + \frac{1}{12}p_2 t^4 + \frac{1}{20}p_3 t^5 \\ v(t) & = & \dot{s}(t) = v_0 + p_0 t + \frac{1}{2}p_1 t^2 + \frac{1}{3}p_2 t^3 + \frac{1}{4}p_3 t^4 \\ \alpha(t) & = & \ddot{s}(t) = p_0 + p_1 t + p_2 t^2 + p_3 t^3 \\ jerk(t) & = & \dddot{s}(t) = p_1 + 2p_2 t + 3p_3 t^2 \end{array} \tag{4.12}$$

where $p_0 = \alpha(0) = \alpha_0$, and $p_1 = jerk(0) = jerk_0$.

The latter is given as:

$$\begin{array}{lcl} s(t) & = & s_0 + v(t_0)t + \frac{1}{2}p_0 t^2 \\ v(t) & = & \dot{s}(t) = v_0 + p_0 t \\ \alpha(t) & = & \ddot{s}(t) = p_0 \\ jerk(t) & = & \dddot{s}(t) = 0 \end{array} \tag{4.13}$$

where $p_0 = \alpha(0) = \alpha_0$.

The boundary conditions shared by these acceleration profiles are:

$$\begin{array}{lcl} jerk_0 & = & jerk(t_0) = p_1 = 0 \\ jerk_1 & = & jerk(t_f) = 0 \\ \alpha_0 & = & \alpha(t_0) \\ v_0 & = & v(t_0) \\ s_0 & = & s(t_0) = 0 \end{array} \tag{4.14}$$

where $t_0$ and $t_f$ refer to the starting and ending time of the acceleration polynomial respectively. and $t_0 = 0$. It should be pointed out that for the profiles adopted by trajectories from the vehicle to the state lattice, $jerk_0$ is equal to the jerk of the vehicle at the start of the planning horizon which might turn out not to be zero. In the cases where $jerk_0 \neq 0$, the resultant profiles can be regarded as a segment of the ones illustrated above.

As $p_0$ and $p_1$ are always given by $\alpha_0$ and $jerk_0$ , only $p_2, p_3$ and $t_f$ remain unknown for the acceleration cubic polynomial of Equation 4.12, requiring two more boundary condition equations ( as $jerk_1 = jerk(t_f) = 0$ already serves as one.) With respect

to the acceleration constant of Equation 4.13, solving for the unknown $t_f$ needs one additional boundary constraint.

In the following, the application purposes of the acceleration profiles and the additional boundary conditions necessary for solving for their unknown parameters are presented. Besides, the approach to verify the feasibility of the acceleration profiles with regard to the speed and acceleration limits of the vehicle is also discussed.

### 4.2.1 Profiles for Acceleration Transition

The acceleration transition profiles are used to generate a smooth change from one acceleration to another. They precede the appearance of a constant acceleration, as is shown in Figure 4.1. They are expressed in the form of acceleration cubic polynomials. The two boundary conditions in addition to those listed in Equation 4.14 required in order to solve the unknown parameters are:

$$\begin{aligned} \alpha_1 &= \alpha(t_f) \\ t_f &= k_{trans}|\alpha_1 - \alpha_0|. \end{aligned} \tag{4.15}$$

where $k_{trans}$ is a predefined constant which can be identified by examining the actual duration that is necessary for a vehicle system (i.e., vehicle dynamics plus controller) to change from one acceleration to another. With the help of these boundary conditions, the closed-form solutions for $p_2$ and $p_3$ can be readily obtained.

The physical capability of the vehicle defines the upper and lower boundaries of possible accelerations and speeds. The limits are denoted as $\alpha_{max}$, $\alpha_{min}$, $v_{max}$ and $v_{min}$ respectively. Accordingly, checking the feasibility of the acceleration profiles means verifying whether there are accelerations exceeding $\alpha_{max}$ or falling below $\alpha_{min}$ along the profiles. The same applies to the feasibility examination of the speed profiles resulting from the acceleration profiles. If the profiles turn out to be infeasible in terms of either acceleration limit or speed limit or both, they should be discarded immediately without further evaluation. As the vehicle is not allowed to perform reverse manoeuvres in the proposed planner, $v_{min}$ is set to zero.

The jerk and acceleration polynomials are the derivatives of the acceleration and speed polynomials respectively. This specific relationship can be exploited to assist the feasibility validation. For example, the acceleration polynomial gets its local extrema at the zeros of the jerk polynomial, and the time coordinate where the jerk polynomial gets its extremum is the inflection point of the acceleration polynomial. Figure 4.2 demonstrates this phenomenon. The same relationship exists between the acceleration and speed polynomials.

The typical acceleration cubic polynomials and their corresponding speed and jerk profiles are demonstrated in Figure 4.1. The fact that the endpoints of the jerk profile has a value of zero determines that the boundaries of the acceleration profile are both

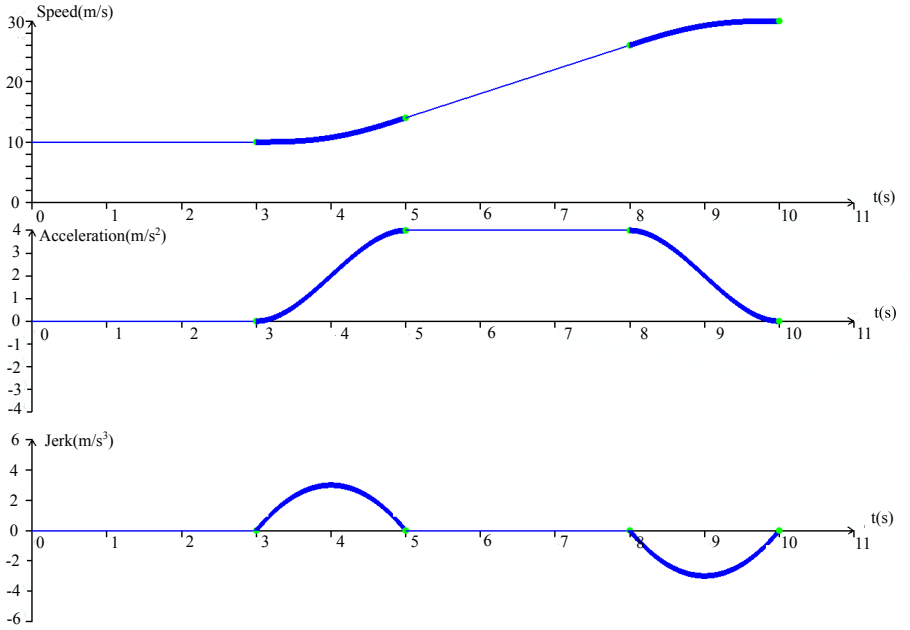



Figure 4.1: Acceleration Profile Examples. The middle, lower and upper graphs show the acceleration profiles and the corresponding jerk and speed changes respectively. On the middle graph, the first bold curve transits the constant acceleration of 0 $m/s^2$ to another constant acceleration of 4 $m/s^2$. On the same graph, the second bold curve transfers the vehicle system to a state where $v = 30\,m/s$ and $\alpha = 0\,m/s^2$.

local extrema. Since the cubic polynomial can have a maximum of two local extrema, the acceleration profile can only fall between the two local extrema, as is shown in Figure 4.2. As a result, the acceleration increases or decreases monotonically within the profile, which means that the acceleration profile is guaranteed to be valid as long as both of its boundaries are within the acceleration limits.

Now it is time to examine the speed profiles. There are generally three cases for consideration. The profiles shown in Figure 4.3(a), Figure 4.3(b), Figure 4.3(c) and Figure 4.3(d) belong to the first case. The feature of this case is that there are no zeros along the acceleration profile except for their endpoints. It can be concluded therefore that there are no local extrema along the speed profiles with their endpoints discounted. Consequently, the speed can only decrease or increase monotonically within the profiles, which indicates that the feasibility of the speed profiles can be guaranteed if both of its endpoints fall within the speed limits.

The second case, as is shown in Figure 4.3(e), is more complicated than the first. As there is one zero along the acceleration profile, a local extremum cannot be avoided along the speed profile, which renders the check relying on the boundary conditions alone insufficient. As the acceleration goes from positive to negative, the local extremum along the speed profile is a local maximum. In the proposed planner, the trajectory with

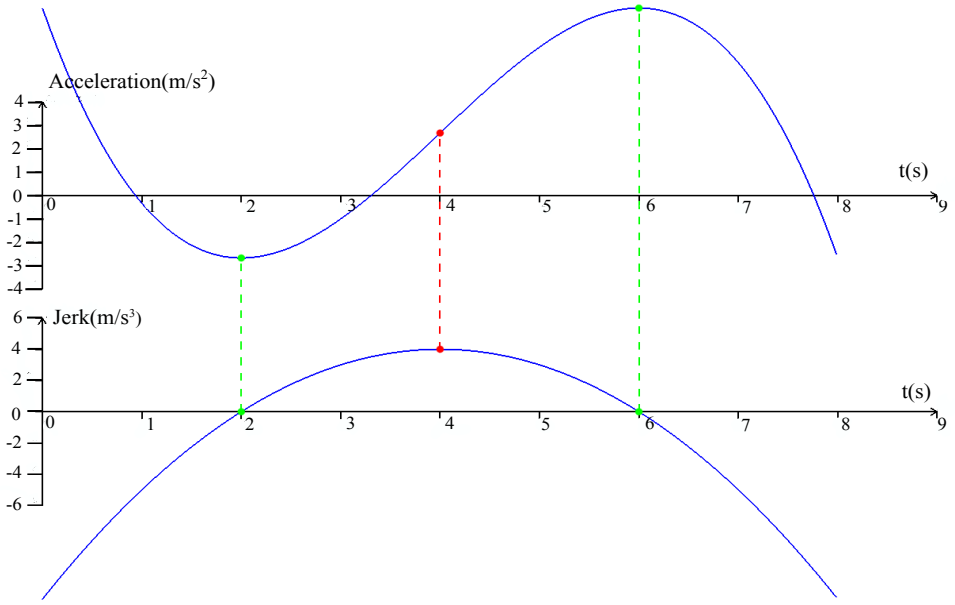
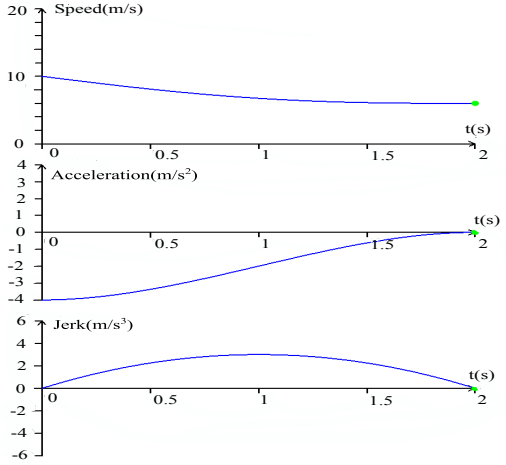


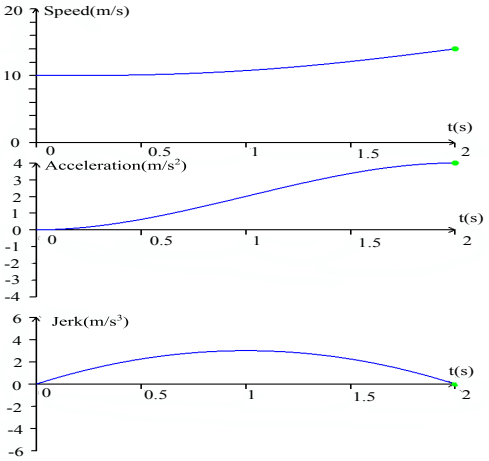

Figure 4.2: Relationship of the jerk and acceleration polynomials. The green points on the jerk curve are the zeros of the jerk polynomial. They correspond to the green points on the acceleration curve where the acceleration polynomial gets its local extrema. The jerk quadratic polynomial gets its extremum at the red point on its curve which is the inflection point of the acceleration cubic polynomial.

speeds exceeding $v_{max}$ is not regarded as infeasible but subject to certain punishments. The reason is twofold. For one thing, $v_{max}$ is usually set to be the speed limit of the current road which is well below the actual potential of the vehicle. Even when $v_{max}$ is set according to the physical limit of the vehicle, it is always a conservative estimation compared to the maximum capability of the vehicle. For another thing, the discrepancy between $v_{max}$ and the possible speeds exceeding $v_{max}$ turns out to be very small in practice if the boundaries of the speed profile are within the limits. As a result, in terms of feasibility validation, the speed profile with a local maximum is only subject to the boundary checking. The trajectory with speeds beyond the speed limits is punished during the evaluation of the dynamic cost.

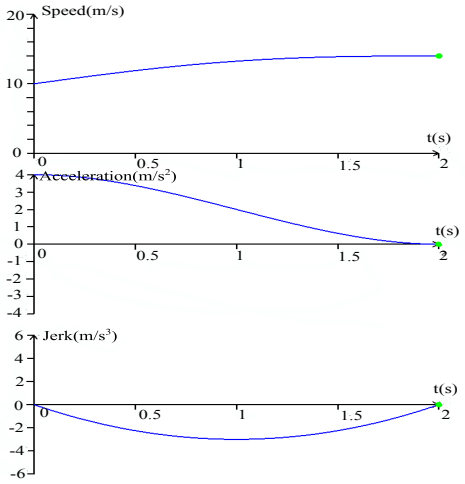
Figure 4.3(f) gives an example of the third case where the local extremum is a local minimum. As speeds less than zero render a trajectory infeasible, it is necessary to check whether this local minimum is negative. As the time coordinates of the local extrema of the speed profile are the zeros of the acceleration profile, to calculate the exact value of the local minimum in question on the speed profile would mean solving for the corresponding root of the acceleration profile equation which is a cubic polynomial equation. Although there are closed-form solutions of the roots of cubic polynomial equations (cf. [83]), it is unnecessarily complicated to implement it in the planner. As



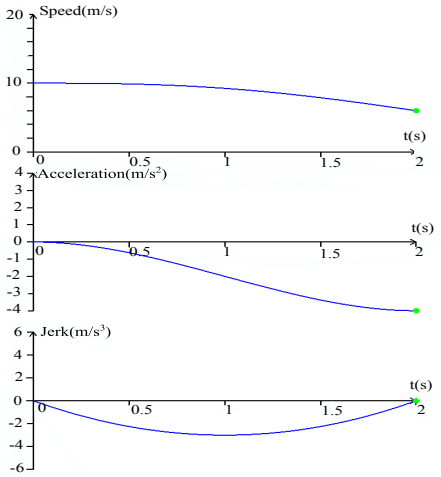

(a) Acceleration increases in the negative acceleration zone without crossing the time axis.



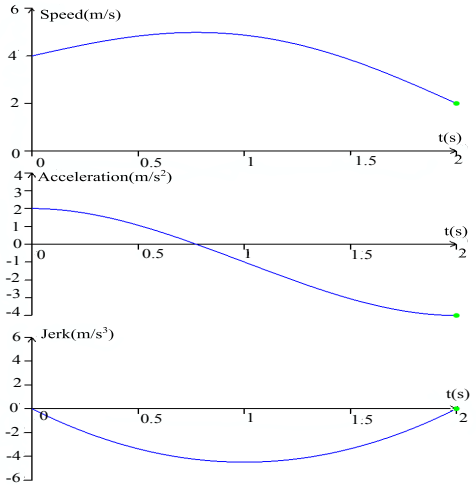

(b) Acceleration increases in the positive acceleration zone without crossing the time axis.



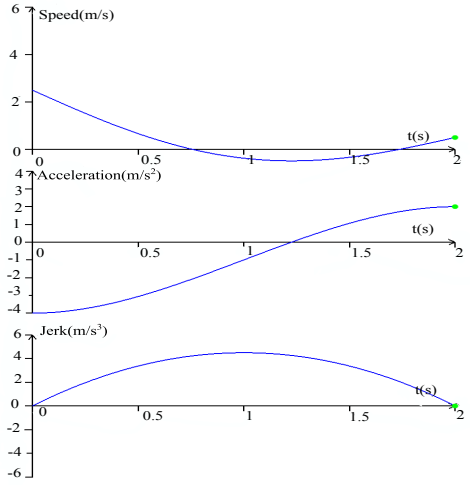

(c) Acceleration decreases in the positive acceleration zone without crossing the time axis.




(d) Acceleration decreases in the negative acceleration zone without crossing the time axis.




(e) Acceleration decreases from the positive acceleration zone to the negative one.




(f) Acceleration increases from the negative acceleration zone to the positive one.

Figure 4.3: The typical acceleration cubic polynomials and their corresponding speed and jerk profiles.

a result, approximate methods are applied to estimate the minimum speed $v_{min}$, which are illustrated as follows:

- Given the acceleration profiles as is shown in Figure 4.4(a), Figure 4.4(b) and Figure 4.4(c), let $t_0, t_1, t_{min}$ refer to the time coordinates where $v_0, v_1, v_{min}$ take place respectively, and let $(t_{inflection}, \alpha_{inflection})$ denote the inflection point of the acceleration polynomial. Note that $t_0, t_1$ are already given by the definition of the acceleration profile. The time coordinate of the axis of symmetry of the jerk polynomial is $t_{inflection}$, i.e., $t_{inflection} = -\frac{p_2}{3p_3}$. Correspondingly, $\alpha_{inflection} = \alpha(t_{inflection})$. Let $t_c$ refer to the time coordinate of the intersection of two specific lines which are $L_1$ that passes through $(t_0, \alpha_0)$ and $(t_1, \alpha_1)$ and $L_2$ that coincides with the time axis. Note that any point that lies on the time axis has an acceleration coordinate of zero. Let $k_c$ represent the slope of $L_1$, and one can have $k_c = (\alpha_1 - \alpha_0)/(t_1 - t_0)$. Let $k_0$ be the slope of the tangent line of the acceleration profile at $t_0$. It follows that $k_0 = jerk(t_0)$. It should be kept in mind that $t_1$ always renders a local extremum (a local maximum in the context of the problem) of the acceleration cubic polynomial as the jerk of the ending state of the profile is always zero. Recall that the jerk of the starting state might not be zero, which might be the case for the trajectory from the vehicle to the state lattice. As a result, $t_0$ might lie to the right of the time coordinate of the other local extremum ( a local minimum in this context). Figure 4.4(a) gives an example of this case. The area of the yellow patch is denoted as $\delta V_r$. It is valid to say that $v_{min} = v_0 - |\delta V_r|$.

- In cases where $k_0 \geq k_c$ as is demonstrated in Figure 4.4(a), let $\delta t$ equal to $\frac{|\alpha_0|}{\alpha_1 - \alpha_0}(t_1 - t_0)$ (i.e., $\delta t = t_c - t_0$). The area of the triangle with the blue frame can be calculated as $\delta V_c = \frac{1}{2}|\alpha_0|\delta t$. Correspondingly, the approximate minimum speed is computed as $v_{min_c} = v_0 - \delta V_c$. Whether $v_{min}$ is negative or not is thus determined by the sign of $v_{min_c}$. It is straightforward that $v_{min_c} < v_{min}$. Therefore it is safe to conclude that $v_{min}$ is positive if $v_{min_c}$ turns out to be positive. However, if $v_{min_c}$ is negative, $v_{min}$ may not be negative. In this sense, the approximate method, while guarantees to discard the infeasible profiles, might also exclude some valid ones.

- If $k_0 < k_c$, and at the same time $\alpha_{inflection} < 0$, as is shown in Figure 4.4(b), one can have that $\delta t_1 = |t_c - t_{inflection}|$ and $\delta t_2 = |t_0 - t_{inflection}|$. Correspondingly, the area composed of the triangle and rectangle with blue frames is given as $\delta V_c = \frac{1}{2}\delta t_1 |\alpha_{inflection}| + \delta t_2 |\alpha_0|$. The rest of the analysis concerning $\delta V_c$ is the same as that of the case illustrated above.

- If $k_0 < k_c$, and at the same time $\alpha_{inflection} \geq 0$, as is shown in Figure 4.4(c), one can have $\delta t = |t_0 - t_{inflection}|$. Correspondingly, the area of the rectangle with blue

frame is given as $\delta V_c = \delta t|\alpha_0|$. The rest of the analysis is the same as that of the first case.

### 4.2.2 Profiles for Acceleration Keeping

Constant acceleration is another profile employed in the proposed planner. Two constant accelerations of different values can be joined via an acceleration transition profile illustrated previously (cf. Figure 4.1). As the jerk stays at zero within the duration of the constant acceleration, it is better to make the jerk at the endpoints of the transition profiles to be zero for the purpose of jerk continuity. Hence the definition in Equation 4.14. Jerk continuity is not addressed in the theory of minimum-jerk trajectory (cf. Section 4.1), as the trajectory discussed therein consists of only one piece, indicating straightforwardly that its jerk is continuous. It is believed in this thesis that it is necessary to keep jerk continuity along the overall trajectory composed of multiple pieces. In this way, infinite *snap*s (the fourth time derivative of position, i.e., the time derivative of jerk) can be avoided, which can further improve the quality of the trajectory.

The additional boundary condition with respect to constant acceleration profiles that is necessary for solving for $t_f$ is given as:

$$s_1 = s(t_f) \tag{4.16}$$

where $s_1$ is always given as the arc length of a whole path edge or a segment of it.

Since $s(t)$ is a quadratic polynomial (as the acceleration is constant), Equation 4.16 is easy to solve. In the proposed planner, constant accelerations are always applied on a complete path edge or on the remaining segment of a path edge on which some acceleration polynomial has already been applied and finished. Accordingly, constant acceleration profiles are supposed to end at the target node of the path edge. In the cases where negative accelerations are applied, it might happen that the vehicle stops before it reaches the target node. That would mean that no real-number solution is available for Equation 4.16. Sometimes it is necessary for the vehicle to stop halfway between the endpoints of a path edge, e.g., in the case of an emergency stop. In this work therefore, the position and time, $s_{1_c}$ and $t_{f_c}$ respectively, where the vehicle stops in such situation is calculated according to

$$\begin{aligned} t_f &= t_0 + \tfrac{v_0}{|\alpha_c|} \\ s_{1_c} &= s_0 + \tfrac{1}{2}|\alpha_c|t^2 \end{aligned} \tag{4.17}$$

where $\alpha_c$ is the constant acceleration. Special nodes called *EXTRA-SAMPLE*s are created in the proposed planner to record the vehicle states resulting from such scenarios.

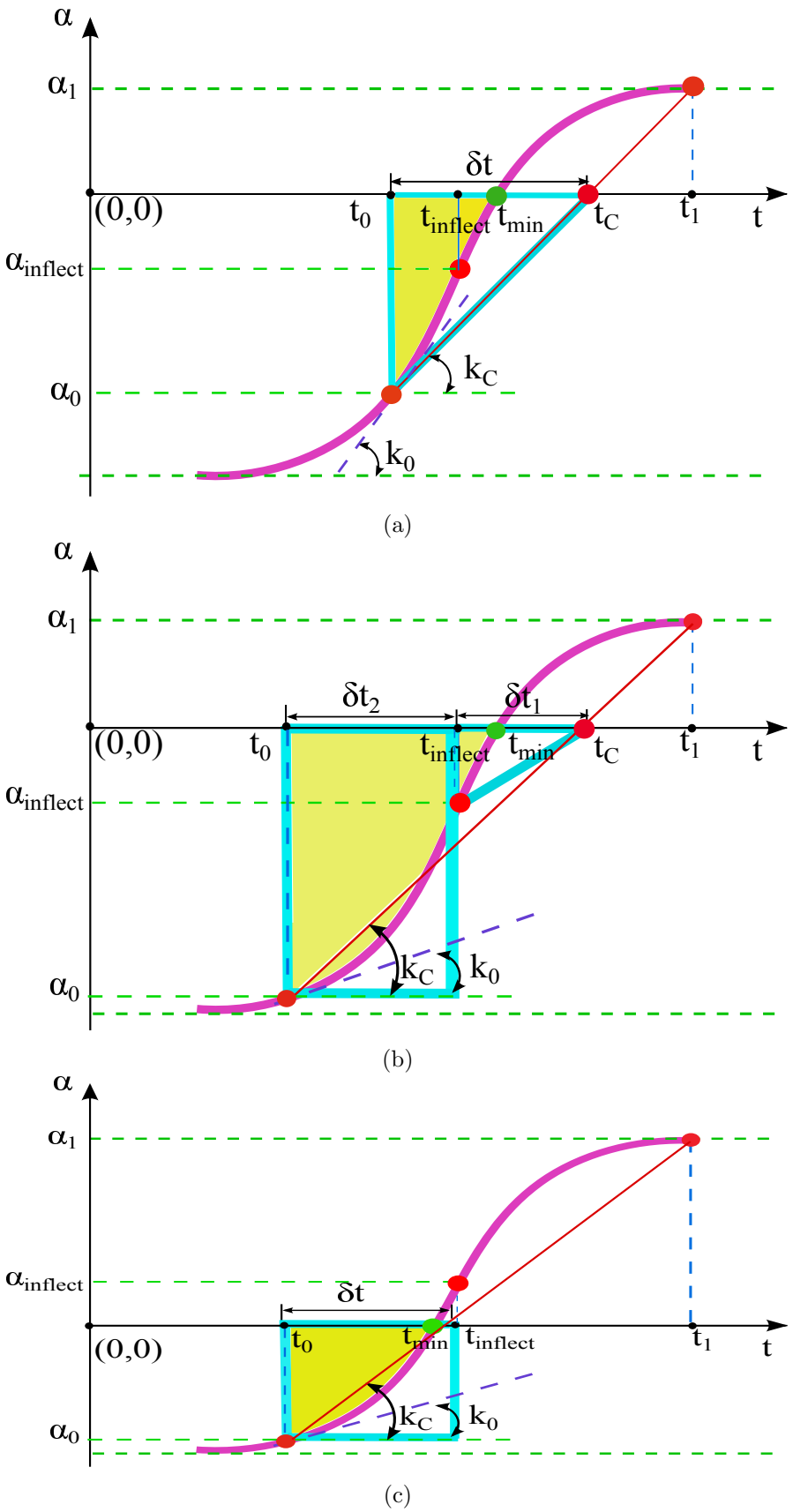



Figure 4.4: Application of approximate methods for estimating the minimum speed in question. The coordinates of some red points are given; those of the other red points can be easily calculated. The time coordinate of the green point is unknown. The red line is $L_1$.

The feasibility of the constant acceleration profile is definite, as the constants are selected among the possible accelerations that are within the limits of the capability of the vehicle. With respect to the feasibility of the corresponding speed profile, only a boundary check is necessary because the speed varies linearly with time.

### 4.2.3 Profiles for Achieving a Target Speed

The vehicle is expected to reach specific speeds in some scenarios such as velocity keeping, stopping, turning and vehicle following. Accordingly, the acceleration profiles for achieving a target speed are indispensable elements of a practical portfolio of acceleration profiles for an on-road motion planner. Such acceleration profiles are in the form of acceleration cubic polynomials, the same as those employed for acceleration transitions, albeit with different boundary conditions.

To design proper boundary conditions for this kind of acceleration profiles, the characteristics of the related traffic scenarios are further investigated. For scenarios such as stopping and making sharp turns, there usually exist specific locations where the vehicle should manoeuvre at the required speed. Ideally, such locations should be incorporated into the set of spatial nodes as long as the planning horizon covers them. The current implementation of the proposed planner does not take into account this issue, which should be improved in future work. Nonetheless, this kind of speeds demanded by these scenarios are categorized as *location-bounded* target speeds in this work. With regard to the scenarios such as velocity keeping, it is necessary for the vehicle to reach a recommended speed (say, a hair below the speed limit of the current road) and maintain that speed for a subsequent period. As a result, it would be more favourable if the acceleration at the end of the transition could be zero. Furthermore, it is not necessary for the vehicle to reach the target speed any specific location in this context. Such target speeds are classified as *acceleration-bounded* target speeds in this thesis.

In addition to the boundary constraints provided in Equation 4.14, more boundary conditions are necessary to solve for the acceleration profiles to achieve *location-bounded* target speeds. They are given as:

$$\begin{aligned} s_1 &= s(t_f) \\ v_1 &= v(t_f) \end{aligned} \tag{4.18}$$

where $s_1$ and $v_1$ are always given. Correspondingly, the additional boundary conditions for solving acceleration profiles aiming at achieving *acceleration-bounded* target speeds are:

$$\begin{aligned} a_1 &= a(t_f) \\ v_1 &= v(t_f) \end{aligned} \tag{4.19}$$

where $a_1$ and $v_1$ are always given.

These boundary conditions in combination with that of $jerk(t_f) = 0$ can be applied to solve for the three unknowns, i.e., $p_2, p_3$ and $t_f$. The fact that $p_1 = jerk_0 = 0$ spares us from having to solve a cubic polynomial equation in the process of calculating the unknowns. As a result, the closed-form solutions of the unknowns can be obtained with ease.

The feasibility of the resultant acceleration cubic polynomials can be verified in the same way as that of the acceleration transition polynomials.

So far, all the types of acceleration profiles applied in the proposed planner are illustrated. It is noteworthy that the vehicle following behaviour has not yet been covered by the presented portfolio of acceleration profiles. Nonetheless, two candidate profiles are recommended here for future work. One is to use the profile composed of acceleration transitions and constant accelerations. The amount of the acceleration can be determined by a reactive control law based on the current and expected velocity and distance of the ego-vehicle relative to the vehicle to follow (cf. [14]). The other candidate profile is based on an estimation of the optimal time when the ego-vehicle should arrive at the next station, taking into account the distance of the vehicle to follow relative to the ego-vehicle at that time. With the time and distance (i.e., the arc length of the path edge) as additional boundary conditions, the acceleration polynomial can be solved. For the current implementation of the proposed planner, the vehicle following behaviour is only assisted by the *follow* area defined behind the vehicle to follow. The ego-vehicle would get penalized if it were to enter the *follow* area (cf. Section 3.5).

## 4.3 Application of Acceleration Profiles

During the graph construction process, the acceleration profiles presented in the previous section are applied on the path edges to generate trajectory edges. The problem of associating the acceleration profiles with the path edges within the state lattice can be formulated as: given a source node $N_0$ and a path edge $\hat{E}_{\hat{N}_0 \to \hat{N}_1}$, apply each of the available acceleration profiles on $\hat{E}_{\hat{N}_0 \to \hat{N}_1}$; should a resultant trajectory edge $E_{N_0 \to N_1}$ turn out to be feasible and traversable, its ending state would become one candidate competing for representing the resultant state lattice node $N_1$ (please refer to Chapter 3 for the meaning of the notations).

The general principles guiding the construction of the trajectories are listed as follows:

**Principle 4.1.** *It might turn out that an acceleration profile cannot end at the target node of the current edge where it is applied. In such case, if the current edge survives the pruning, the unfinished acceleration profile will be continued in the subsequent path edges. The new acceleration profiles that will be associated with the subsequent path edges start at the point where the unfinished profile ends.*

**Principle 4.2.** *One path edge can be used to construct several different trajectory edges, each associated with a unique acceleration profile. If a path edge starting from the lattice node $N_0$ ends up being fully covered by the remaining part of an unfinished acceleration profile that is passed down from previous edges, the new acceleration profiles prepared to be applied on this path edge have no chance at all to play a role. As a result, all the trajectory edges that start from the lattice node $N_0$ and are intended to be constructed by different acceleration profiles would turn out to be the same. Consequently, only one of them is necessary. As the acceleration profile index is used to decode some information of the acceleration profile pertaining to it for the reconstruction of the best trajectory, the trajectory edge that is intended to have the same acceleration profile index as the unfinished one is chosen to be applied.*

**Principle 4.3.** *Should an acceleration profile be completed before reaching $N_1$ ( which can only happen in the case of acceleration profiles for achieving an acceleration-bounded target speed and those for acceleration transition), a constant acceleration with the same value as the ending acceleration of the acceleration profile would be applied on the rest of the path edge.*

In sum, one acceleration profile might extend over several trajectory edges, and one trajectory edge might contain several acceleration profiles.

The algorithms related to the construction of the trajectories within the state lattice are illustrated in Algorithm 1 and Algorithm 2. It is worth mentioning that the speed and time pertaining to the target node of the constructed trajectory edge should always be calculated. The reason is twofold. For one thing, the target node has to compete for representing one time-speed grid cell $c_{i_t i_v}$. Consequently, it needs to know which grid cell it falls into. Should the target node win against other candidates, its information would be stored in the memory segmentation allocated based on its index including $i_t$ and $i_v$, which also requires the time and speed of the target node. For another thing, the desirability of the target node is evaluated based on a heuristic function expressed in terms of the time and speed represented by the node in question (cf. Section 3.4), which is necessary for the pruning operation to decide which candidate node wins over all others.

Both considerations illustrated above require the target node to provide its speed and time. This requirement would mean finding the roots of a quintic polynomial from the perspective of the edges where an applied acceleration cubic polynomial does not end within it, which is hard to realize. To that end, rough estimations of the speed and time which are a by-product of the dynamic cost evaluation are assigned to the target node if it is hard to obtain the exact values. If the sampling step for the dynamic cost evaluation is small enough, the discrepancy between the estimation and the actual value is tolerable. Nonetheless, a more detailed error analysis is necessary which is one task

for future work. Even the error is intolerably large, the only consequences thereby will be that the target node is confronted with unfair judgement by the pruning operation or that it might be stored in a memory segment that is not consistent with its actual index. Note that the rough estimation does not provide any kind of reference for any other operations such as the construction of the subsequent acceleration profiles and the reconstruction of the selected trajectory. The arc length of the edges and the acceleration polynomials are the only references for those operations, which guarantees that the final representation of the trajectory is accurate without any approximation. In addition, as the size of the time-speed grid is limited, the candidate node that falls outside the grid would be assumed to fall within the grid cell that lies nearest to it. Concretely, if the original time-speed index of one node is $(i_t^n, i_v^n)$ which is calculated by dividing its time and speed by the corresponding sampling units, a modified time-speed index $(i_t, i_v)$ that maps the node to the time-speed grid is given as:

$$\begin{aligned} i_t &= min(i_t^n, i_t^{max}) \\ i_v &= min(i_v^n, i_v^{max}) \end{aligned} \tag{4.20}$$

where $i_t^{max}$ and $i_v^{max}$ represent the maximum time and speed indices of the grid respectively. The pruning and storage operations would be conducted based on the modified index.

In addition, a constant acceleration must be preceded by an acceleration transition profile if the ending acceleration of the preceding profile is not equal to this constant acceleration. The resultant profile composed of the transition and constant acceleration profiles are regarded as one acceleration profile and share a common acceleration profile index.

The algorithms are implemented on the CUDA-enabled graphics hardware so that the construction of the trajectory edges starting from the nodes at the same station can be carried out in parallel. After the graph construction is finished, the best constraint-abiding trajectory would be regenerated by tracing back from the best target node to the starting state of the vehicle. This process is implemented sequentially on the CPU. More implementation details can be found in Chapter 5.

In comparison to the construction of the trajectory edges within the state lattice, the generation of the trajectory edges from the vehicle to the state lattice needs some special treatments in order to guarantee a certain level of consistency between the trajectories generated from consecutive planning horizons. It is assumed in the proposed planner that the vehicle executes the control commands correctly. In the current implementation therefore, the jerk of the vehicle at the start of each planning cycle (denoted as $jerk_{init}$) is extracted from the preceding trajectories. As $jerk_{init}$ might not be the starting or ending jerk of a complete jerk profile, it might not be equal to zero. This disobeys

the boundary condition which defines that $jerk_0 = 0$ for all acceleration profiles, which causes some inconveniences in the application of acceleration profiles. The fact that $jerk_0 \neq 0$ means that $p_1 \neq 0$. A non-zero $p_1$ might make it necessary to solve for the roots of a cubic polynomial equation in the cases where acceleration cubic profiles for achieving a target speed are applied. To avoid the complicated computation, only transition-constant acceleration profiles are allowed to be applied on the trajectory edges from the vehicle to the state lattice when $jerk_{init} \neq 0$. Besides, it is more encouraged to continue the acceleration profile from which $jerk_{init}$ is extracted in order to maintain a certain level of planning consistency between consecutive planning cycles.

Figure **??** shows typical strategies in the association process.

## 4.4 Evaluation

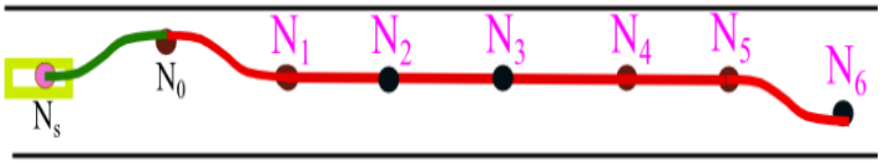
With the acceleration profiles, their association with the path edges and the evaluation of their feasibility and traversability presented, this section evaluates the performance of this trajectory representation strategy. In the following, the concrete parameters of the acceleration profiles applied in the proposed planner are first provided. After that, the performance evaluation is reported.

### 4.4.1 Concrete Acceleration Profiles Applied in the Planner

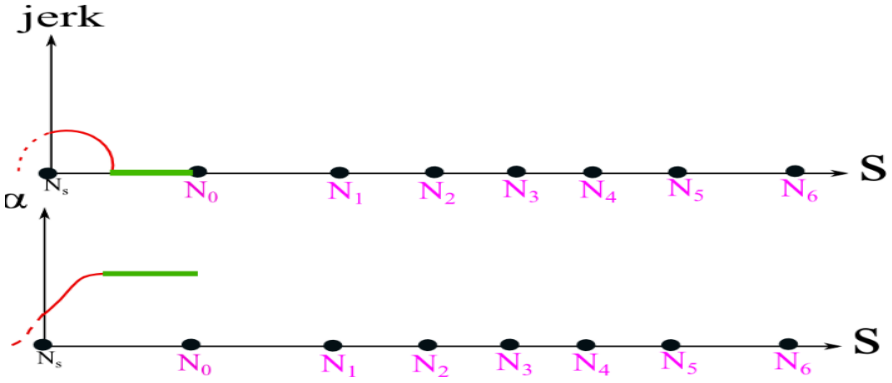
The concrete acceleration profiles employed in the proposed planner are listed in Table 4.1. It should be pointed out that a larger number of acceleration profiles are possible, with the only constraint being the requirement on the computational efficiency. Furthermore, the different target-speed acceleration profiles may not necessarily be present at the same time, as it is common that the target speed for a specific traffic scenario be unique. For example, if the vehicle is allowed to reach the speed limit of the road, it would be unnecessary for it to keep an eye on the trajectories with a target speed for turning. Nonetheless, for the sake of safety, the acceleration profile with the target speed of zero is always included in the acceleration profile portfolio for all kinds of traffic scenarios.

### 4.4.2 Performance Evaluation

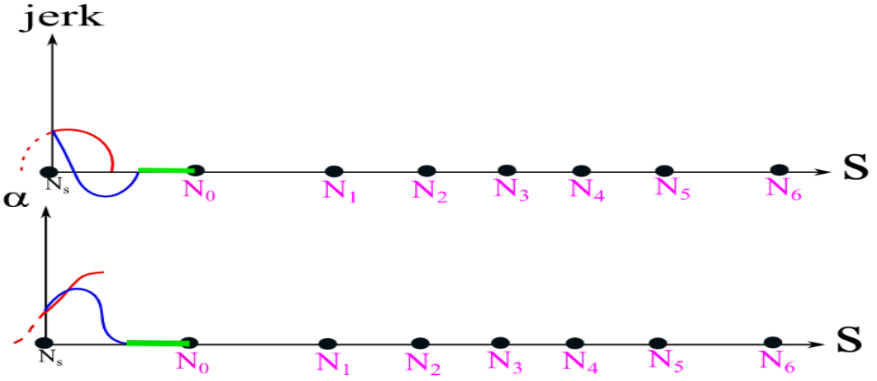
As the presented acceleration profiles and their application strategy are intended to generate trajectories with a high level of smoothness, this evaluation focuses on trajectory smoothness. The benchmarks adopted here are the jerk levels of the position quintic polynomials which have the same position, speed and acceleration boundary conditions as the trajectories generated by the proposed motion planner. Such position quintic polynomials are specified as *uniform polynomials* in this section in the sense that they are not composed of several trajectory segments as opposed to the *multi-piece* trajectories, i.e., the trajectories generated by the proposed search-based planner. Three typical vehicle motions are selected for the evaluation. Among them are accelerating from stop,



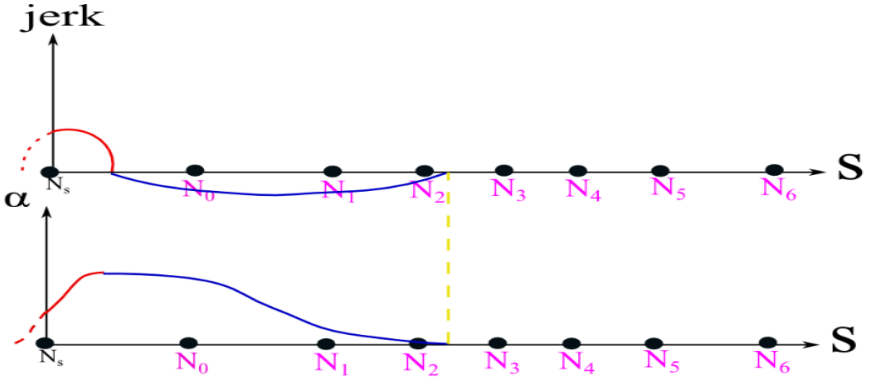

(a) A path (red curve). The association starts from the path edge of $N_sN_0$.




(b) There is an unfinished profile (dashed red +solid red) from the last plan. Complete the unfinished profile (solid red). if $N_0$ has not been reached, then apply constant acceleration (green)




(c) There is an unfinished profile (dashed red +solid red) from the last plan.We are allowed to apply constant acclerations (green) from the starting point without completing the unfinished part firstly. However, if the to-be-applied constant acceleration has a different value than the starting point, an accleration transition (blue) is necessary.




(d) There is an unfinished profile (dashed red +solid red) from the last plan.Complete the unfinished profile (solid red). Then an accelerarion profile (blue) for reaching a target speed is applied. However, this profile cannot end at $N_0$. As a result, it expands over several path edges.

Figure 4.5: Typical association strategies.

**Algorithm 1** TRAJECTORY-EDGE-CONSTRUCTION

**Input:** $n_0 : (v_0^n, t_0^n, cost_0^{traverse})$ // *Information of source node $n_0$ including traversal cost from vehicle to $n_0$.*

**Input:** $P_{last} : (t_0^{last}, v_0^{last}, t_D^{last}, p_0^{last}, p_1^{last}, p_2^{last}, p_3^{last}, s_{remain}^{last})$ // *Initial time and speed, duration, cubic polynomial coefficients and unfinished distance of acceleration profile $P_{last}$ ending at, or passing, or ending in front of $n_0$ without any other cubic polynomial between them.*

**Input:** $i_\alpha^0$ // *Index of acceleration profile $P_{last}$.*

**Input:** $i_t^{last}, \delta t_{last}, \kappa_{last}$ // *Sampling information of edge ending at $n_0$ for dynamic cost evaluation.*

**Input:** $\rho_{cur} : (a_0, a_1, a_2, a_3, x_0, y_0, \theta_0, s_\rho), cost_{static}$ // *Model and static cost of path edge.*

**Input:** $\{i_\alpha\}$ // *Acceleration profile indices.*

**Output:** $\{n_1\} : \{(v_1^n, t_1^n, cost_1^{traverse})\}$ // *Candidates for target node.*

**Output:** $\{P_{cur}\} : \{(t_0^{cur}, v_0^{cur}, t_D^{cur}, p_0^{cur}, p_1^{cur}, p_2^{cur}, p_3^{cur}, s_{remain}^{cur})\}$ // *Acceleration profiles for trajectories in update.*

**Output:** $\{(i_t^{cur}, \delta t_{cur}, \kappa_{cur})\}$ // *Sample information of edges in update for dynamic cost evaluation.*

**for all** $i_\alpha \in \{i_\alpha\}$ **do**
  FLAG-EXTRA-SAMPLE$\leftarrow$ FALSE // *Flag indicating whether acceleration profile stops halfway along path edge.*
  FLAG-CONSTANT $\leftarrow$ FALSE // *Flag indicating whether there is constant acceleration profile along path edge.*
  **if** $s_{remain}^{last} \geq s_\rho$ **then**
    **if** $i_\alpha \neq i_\alpha^0$ **then**
      continue // *Follow Principle 4.2.*
    **end if**
  **end if**
  **if** $s_{remain}^{last} \geq s_\rho$ **then**
    $(t_0^{cur}, v_0^{cur}, t_D^{cur}, p_0^{cur}, p_1^{cur}, p_2^{cur}, p_3^{cur}) \leftarrow (t_0^{last}, v_0^{last}, t_D^{last}, p_0^{last}, p_1^{last}, p_2^{last}, p_3^{last})$ // *Apply the remaining part of the unfinished acceleration profile.*
  **else**
    $\alpha_0^{cur} \leftarrow$CUBIC-POLY$(p_0^{last}, p_1^{last}, p_2^{last}, p_3^{last}, t_D^{last})$
    **if** $s_{remain}^{last} > 0$ **then** // *Last profile is unfinished but will end on $\rho_{cur}$.*
      Initial arc length of profile $P_{cur}$: $s_0^{cur} \leftarrow s_{remain}^{last}$
      Initial speed of profile $P_{cur}$: $v_0^{cur} \leftarrow$ QUARTIC-POLY$(v_0^{last}, t_D^{last}, p_0^{last}, p_1^{last}, p_2^{last}, p_3^{last})$
      Initial acceleration of profile $P_{cur}$: $\alpha_0^{cur} \leftarrow$ CUBIC-POLY$(p_0^{last}, p_1^{last}, p_2^{last}, p_3^{last}, t_D^{last})$ // *Calculate ending speed and acceleration of profile $P_{last}$.*
      $t_0^{cur} \leftarrow t_D^{last} + t_0^{last}$
    **else**
      $s_0^{cur} \leftarrow 0$
      $(v_0^{cur}, \alpha_0^{cur}, t_0^{cur}) \leftarrow (v_0^n, \alpha_0^n, t_0^n)$
    **end if**
    $p_0^{cur} \leftarrow \alpha_0^{cur}$
    $p_1^{cur} \leftarrow 0$

**Algorithm 1** TRAJECTORY-EDGE-CONSTRUCTION (continued)

```
24:     if i_α refers to transition-constant acceleration profile OR
            acceleration-bounded target-speed profile then
25:         α_1^cur ← α(i_α)
26:         if i_α refers to transition-constant acceleration profile AND α_1^cur ≡ α_0^cur
            then
27:             p_2^cur ← 0
28:             p_3^cur ← 0
29:             t_D^cur ← 0
30:             s_D^cur ← 0
31:             Arc length of constant acceleration s_D^const ← s_ρ − s_0^cur
32:             Initial speed of constant acceleration v_0^const ← v_0^cur
33:             Initial time of constant acceleration t_0^const ← t_0^cur
34:             FLAG-CONSTANT← TRUE
35:         else
36:             if i_α refers to transition-constant acceleration profile then
37:                 t_D^cur ← k_trans|α_1^cur − α_0^cur|                    // cf. Section 4.2.
38:                 (p_2^cur, p_3^cur, s_D^cur, v_1^cur)        ←TRANSITION-PROFILE-
                    GENERATION (α_1^cur, t_D^cur, v_0^cur, α_0^cur, p_0^cur, p_1^cur)
                                    // Generate transition profile (cf. Section 4.2).
39:             else                    // Acceleration-bounded target-speed profile.
40:                 v_1^cur ← v(i_α)
41:                 (p_2^cur, p_3^cur, s_D^cur, t_D^cur) ← ACCEL-BOUNDED-PROFILE-
                    GENERATION(α_1^cur, v_1^cur, v_0^cur, α_0^cur, p_0^cur, p_1^cur)
                // Generate acceleration-bounded target-speed profile (cf. Section 4.2).
42:             end if
43:             if s_D^cur < s_ρ − s_0^cur then //  Acceleration profile ends before reaching
                                                    target node.
44:                 s_D^const ← s_ρ − s_D^cur − s_0^cur
45:                 t_0^const ← t_0^cur + t_D^cur
46:                 v_0^const ← v_1^cur
47:                 FLAG-CONSTANT ← TRUE            // Follow Principle 4.3.
48:             end if
49:         end if
50:     else                        // i_α refers to location-bounded target-speed profile.
51:         v_1^cur ← v(i_α)
52:         s_D^cur ← s_ρ − s_0^cur
53:         (p_2^cur, p_3^cur, t_D^cur, α_1^cur) ← LOCATION-BOUNDED-PROFILE-
            GENERATION(s_D^cur, v_1^cur, v_0^cur, α_0^cur, p_0^cur, p_1^cur)
                // Generate acceleration-bounded target-speed profile (cf. Section 4.2).
54:     end if
55: end if
56: if ACCELERATION-SPEED-FEASIBLE(P_cur) returns FALSE then
                        //  Verify acceleration-speed feasibility of P_cur (cf. Section 4.2).
57:     Continue
58: end if
59: if FLAG-CONSTANT ≡ TRUE then            // Apply constant acceleration.
60:     (v_1^n, t_1^n, s_D^const(real), t_D^const)  ←  CONSTANT-ACCELERATION-
        APPLICATION(v_0^const, α_1^cur, s_D^const, t_0^const)
                                    //  Apply constant acceleration ( cf. Section 4.2).
```

**Algorithm 1** TRAJECTORY-EDGE-CONSTRUCTION (continued)

**if** $s_D^{const}(real) < s_D^{const}$ **then** // *Profile stops before reaching target node.*
FLAG-EXTRA-SAMPLE $\leftarrow$ TRUE // *cf. Section 4.2.*
$s_D^{const} = s_D^{const}(real)$
**end if**
**end if**
$N_s \leftarrow 0$ // Record #samples for dynamic cost evaluation.
$cost_1^{traverse} \leftarrow 0$
$\theta_1 \leftarrow \theta_0$
$x_1 \leftarrow x_0$
$y_1 \leftarrow y_0$
$s_1 \leftarrow 0$
**if** $s_{remain}^{last} > 0$ **then** // *Evaluate unfinished segment of* $P_{last}$.
$s_D^{last} \leftarrow$QUINTIC-POLY$(v_0^{last}, p_0^{last}, p_1^{last}, p_2^{last}, p_3^{last}, t_D^{last})$
Arc length of evaluated segment $s_{past} \leftarrow s_D^{last} - s_{remain}^{last}$
// *Positive for evaluated segment preceding current path edge, negative for evaluated segment on current path edge.*
Time sampling step for dynamic cost evaluation $\delta t \leftarrow \delta t_{last}$
Dynamic cost evaluation ends at arc length $s_{eval} \leftarrow min(s_{remain}^{last}, s_\rho)$
$(cost_{dynamic}, i_t^{cur}, \kappa_{cur}, \theta_1, x_1, y_1, s_1) \leftarrow$
DYNAMIC-COST$(P_{last}, \rho, i_t^{last}, \delta t, s_{eval}, s_{past}, \kappa_{last}, \theta_1, x_1, y_1, s_1)$
// *Dynamic cost evaluation (cf. Algorithm 2).*
**if** $cost_{dynamic} \equiv \infty$ **then**
continue // *Discard untraversable trajectory.*
**end if**
$cost_1^{traverse} \leftarrow cost_1^{traverse} + cost_{dynamic}$
$N_s \leftarrow N_s + i_t^{cur} - i_t^{last}$
**if** $s_{remain}^{last} > s_\rho$ **then** // *Estimate speed and time at target node for unfinished profile (cf. Section 4.3).*
Estimated duration of evaluated segment of last profile $t_D \leftarrow i_t^{cur}\delta t$
$t_1^n \leftarrow t_0^{last} + t_D$
$v_1^n \leftarrow$ QUARTIC-POLY $(v_0^{last}, t_D, p_0^{last}, p_1^{last}, p_2^{last}, p_3^{last})$
$s_{remain}^{cur} \leftarrow s_{remain}^{last} - s_\rho$
**else if** $s_{remain}^{last} \equiv s_\rho$ **then**
$t_1^n \leftarrow t_0^{last} + t_D^{last}$
$v_1^n \leftarrow$ QUARTIC-POLY$(v_0^{last}, t_D^{last}, p_0^{last}, p_1^{last}, p_2^{last}, p_3^{last})$
$s_{remain}^{cur} \leftarrow 0$
**end if**
**end if**
**if** $(s_{remain}^{last} < s_\rho)$ **AND** $(t_D^{cur} > 0)$ **then** // *Evaluate new polynomial profile.*
$s_{past} \leftarrow -s_0^{cur}$
$N_t^{cur} \leftarrow s_D^{cur}$/PATH-SAMPLING-STEP+1
$\delta t \leftarrow t_D^{cur}/N_t^{cur}$
$i_t^{cur} \leftarrow 1$
$s_{eval} \leftarrow min(s_0^{cur} + s_D^{cur}, s_\rho)$
$N_s \leftarrow N_s - i_t^{cur}$
$(cost_{dynamic}, i_t^{cur}, \kappa_{cur}, \theta_1, x_1, y_1, s_1) \leftarrow$
DYNAMIC-COST$(P_{cur}, \rho, i_t^{cur}, \delta t, s_{eval}, s_{past}, \kappa_{cur}, \theta_1, x_1, y_1, s_1)$
**if** $cost_{dynamic} \equiv \infty$ **then**
continue
**end if**

**Algorithm 1** TRAJECTORY-EDGE-CONSTRUCTION (continued)

105: $cost_1^{traverse} \leftarrow cost_1^{traverse} + cost_{dynamic}$
106: $N_s \leftarrow N_s + i_t^{cur}$
107: **if** $s_0^{cur} + s_D^{cur} > s_\rho$ **then**
108: Estimated duration of evaluated segment of current profile $t_D \leftarrow i_t^{cur}\delta t$
109: $t_1^n \leftarrow t_0^{cur} + t_D$
110: $v_1^n \leftarrow$ QUARTIC-POLY$(v_0^{cur}, t_D, p_0^{cur}, p_1^{cur}, p_2^{cur}, p_3^{cur})$
111: $s_{remain}^{cur} \leftarrow s_0^{cur} + s_D^{cur} - s_\rho$
112: **else if** $s_0^{cur} + s_D^{cur} \equiv s_\rho$ **then**
113: $t_1^n \leftarrow t_0^{cur} + t_D^{cur}$
114: $v_1^n \leftarrow v_1^{cur}$
115: $s_{remain}^{cur} \leftarrow 0$
116: **end if**
117: **end if**
118: **if** FLAG-CONSTANT≡ TRUE **then**
*// Evaluate segment with constant acceleration.*
119: $s_{past} \leftarrow -(s_0^{cur} + s_D^{cur})$
120: $N_t^{cur} \leftarrow s_D^{const}$/PATH-SAMPLING-STEP+1
121: $\delta t \leftarrow t_D^{const}/N_t^{cur}$
122: $i_t^{cur} \leftarrow 1$
123: $s_{eval} \leftarrow s_0^{cur} + s_D^{cur} + s_D^{const}$
124: $N_s \leftarrow N_s - i_t^{cur}$
125: $P_{const} \leftarrow (t_0^{const}, v_0^{const}, t_D^{const}, \alpha_1^{cur}, 0, 0, 0, 0)$
126: $(cost_{dynamic}, i_t^{cur}, \kappa_{cur}, \theta_1, x_1, y_1, s_1) \leftarrow$
DYNAMIC-COST$(P_{const}, \rho, i_t^{cur}, \delta t, s_{eval}, s_{past}, \kappa_{cur}, \theta_1, x_1, y_1, s_1)$
127: **if** $cost_{dynamic} \equiv \infty$ **then**
128: continue
129: **end if**
130: $cost_1^{traverse} \leftarrow cost_1^{traverse} + cost_{dynamic}$
131: $N_s \leftarrow N_s + i_t^{cur}$
132: $s_{remain}^{cur} \leftarrow 0$
133: **end if**
134: $cost_1^{traverse} \leftarrow cost_1^{traverse} s_\rho / N_s$ *// Normalize dynamic cost by steps/$s_\rho$ so that dynamic cost does not depend on $N_s$.*
135: $cost_1^{traverse} \leftarrow cost_1^{traverse} + cost_0^{traverse} + cost_{static}$+OTHER-TRAVERSE-COST
*// See Subsection 5.2.4 for OTHER-TRAVERSE-COST.*
136: Desirability of target node
$cost_{target} \leftarrow$COST-SPEED-TIME-AS-TARGET$(v_1^n, t_1^n)$
*// See Subsection 5.2.3 for desirability evaluation.*
137: $cost = cost_1^{traverse} + cost_{target}$
138: Time-speed grid cell index $i_t, i_v \leftarrow$ GRID-INDEX$(t_1^n, v_1^n)$ *// See Equation 4.20.*
139: Target node index $i_n^1 \leftarrow (i_s, i_l, i_\alpha, i_t, i_v)$
140: **if** SUCCEED-IN-PRUNING$(i_n^1, cost)$ **then**
141: $\delta t_{cur} = \delta t$
142: $SAVE(i_n^1, n_1, P_{cur}, i_t^{cur}, \delta t_{cur}, \kappa_{cur})$
*// Save trajectory information at memory allocation designated by $i_n^1$.*
143: **end if**
144: **end for**

**Algorithm 2** DYNAMIC-COST($P, \rho, i_t^0, \delta t, s_{eval}, s_{past}, \kappa_0, \theta_0, x_0, y_0, s_0$)

*// See Algorithm 1 for the meaning of notations.*

**Input:** DYNAMIC-COST-MAP, $P, \rho, i_t^0, \delta t, s_{eval}, s_{past}, \kappa_0, \theta_0, x_0, y_0, s_0$

**Output:** $cost_{dynamic}, i_t^1, \kappa_1, \theta_1, x_1, y_1, s_1$

$cost_{dynamic} \leftarrow 0$
$i_t^1 \leftarrow i_t^0$
$t \leftarrow i_t^1 \delta t$
$s_1 \leftarrow$ QUINTIC-POLY $(P.v_0, P.p_0, P.p_1, P.p_2, P.p_3, t) - s_{past}$
**while** $s_1 \leq s_{eval}$ **do**
  $v_1 \leftarrow$QUARTIC-POLY$(P.v_0, t, P.p_0, P.p_1, P.p_2, P.p_3)$
  $\alpha_1 \leftarrow$CUBIC-POLY$(P.p_0, P.p_1, P.p_2, P.p_3, t)$
  $(\kappa_1, \theta_1) \leftarrow$ CURVATURE-HEADING $(\rho.a_0, \rho.a_1, \rho.a_2, \rho.a_3, \rho.\theta_0, s_1)$
    *// Calculate curvature and heading at point $s_1$ along $\rho$.*
  $x_1 \leftarrow x_0 + (cos(\theta_1) + cos(\theta_0))(s_1 - s_0)/2$  *// Estimate the position using trapezoidal integration.*
  $y_1 \leftarrow y_0 + (sin(\theta_1) + sin(\theta_0))(s_1 - s_0)/2$
  $cost_{obstacle} \leftarrow$ DYNAMIC-COST-MAP$(x_1, y_1, P.t_0 + t)$
  $cost_{states} \leftarrow$DYNAMIC-FEASIBILITY-COST$(v_1, \alpha_1, \kappa_1, \kappa_0, \delta t)$
    *// See Subsection 5.2.4 for DYNAMIC-FEASIBILITY-COST.*
  $cost_{dynamic} \leftarrow cost_{obstacle} + cost_{states} + cost_{dynamic}$
  $\theta_0 \leftarrow \theta_1$
  $\kappa_0 \leftarrow \kappa_1$
  $x_0 \leftarrow x_1$
  $y_0 \leftarrow y_1$
  $s_0 \leftarrow s_1$
  $i_t^1 \leftarrow i_t^1 + 1$
  $t \leftarrow i_t^1 \delta t$
  $s_1 \leftarrow$ QUINTIC-POLY$(P.v_0, P.p_0, P.p_1, P.p_2, P.p_3, t) - s_{past}$
**end while**

| Type | | Value | Description |
|---|---|---|---|
| Acceleration constant | | $\alpha_{brake}^{soft} = -2\ m/s^2$ | Maximum comfortable braking |
| | | $\alpha_{brake}^{hard} = -4\ m/s^2$ | Maximum braking allowed with penalty |
| | | $\alpha_{gas}^{soft} = 1\ m/s^2$ | Maximum comfortable acceleration |
| | | $\alpha_{gas}^{hard} = 2\ m/s^2$ | Maximum acceleration allowed with penalty |
| | | $\alpha = 0\ m/s^2$ | Velocity keeping |
| Acceleration transition | | $k_{trans} = 0.5, 1, 2$ | From given acceleration $\alpha_0$ to any of the constant accelerations $\alpha_1$ that are listed above |
| Target speed | Location-bounded | $v_1 = 0\ m/s$ | Stop |
| | | $v_1 = 1\ m/s$ | Low speed for hard turning |
| | Acceleration-bounded | $v_1 = 0.99\ v_{limit}$, $\alpha_1 = 0\ m/s^2$ | Smooth transition to the state of velocity keeping at a hair below the speed limit |

Table 4.1: Acceleration profiles applied in the proposed planner.

decelerating from high speed and a combination of decelerating and accelerating. The first motion is a common behaviour of the vehicle. The second one can be necessary if, for example, the vehicle is to stop in front of a traffic light. The last one is useful when the vehicle makes a relatively hard turning manoeuvre. For the first and the third motions, only the trajectories generated in one planning cycle are evaluated. With regard to the second motion, the actual trajectory of the vehicle from the moment when the stopping signal is assigned to the time when the vehicle gets full stop is recorded and evaluated. This special treatment for the second motion is necessary because the short planning horizon ($100\,m$) is not sufficient for the vehicle at a speed of $30\,m/s$ to stop within it (as the hardest deceleration is set to be $-4\,m/s^2$ ). The configuration of the parameters and the evaluation result are provided in Table 4.2. The performance of the proposed trajectory representation strategy from the perspective of the reported evaluation can be summarised as follows:

- The jerk levels of the multi-piece trajectories tend to be higher than that of the uniform polynomials. As can be seen from the graphs of jerk, the uniform polynomials distribute the jerk across its duration; by contrast, the multi-piece trajectories finish the necessary acceleration or deceleration within one small segment of its overall duration, which causes relatively large jerk, resulting in comparatively large jerk level. The reason why it is hard for the multi-piece trajectories to behave in the same way as the uniform trajectories is twofold. For one thing, the number of the acceleration profiles available to the planner is rather limited. To be more concrete, the available acceleration profiles are not enough for the multi-piece trajectories to approximate the behaviour of the uniform polynomials to a satisfactory extent. For another thing, the jerk level has not been incorporated into the criteria for the trajectory evaluation. In other words, there might exist candidate trajectories with smaller jerk level than the selected one. Nonetheless, without enriching the portfolio of the acceleration profiles, introducing the criterion of jerk level alone cannot make much difference.

- While the performance demonstrated by the multi-piece trajectories in the first and third scenarios are still satisfying, the jerk level of the multi-piece trajectory generated for the vehicle to come to a stop from running at a relatively high speed is much worse than that of its benchmark. Two causes contribute to this inconsistency in performance. Firstly, for the stopping behaviour, specific evaluation criteria are designed to take care of the target speed and the target location where the vehicle is supposed to have the target speed. Currently, the criteria define that any candidate for the target node that falls on the area in front of the target location must have a speed of zero. Besides, they are fined with extra costs in proportion to their distances relative to the target location. It is also stipulated

that the target nodes that have not yet passed the target location must be able to reach the target location at the target speed according to a rough estimation. These additional cost functions together with other existing cost functions still require further tuning in order to obtain better performance. Furthermore, it would be much better if the lattice could incorporate the target location as one node, as is discussed previously. Secondly, the stopping trajectory does not come from a single planning cycle; instead, it is the trajectory of the vehicle decelerating to a stop. As a result, the stopping trajectory is composed of several pieces, with each piece being generated by different planning cycles. Although the consistency of successive plans is taken into account in the trajectory evaluation criteria, its effect might not be sufficient in this scenario.

- Although the uniform polynomials have a better performance than the multi-piece trajectories in terms of jerk level, it cannot be easily applied. It seems to be easy to implement them in the evaluation because the target states are always assumed to be given, i.e., the target states are directly obtained from the multi-piece trajectories with which they are compared. In real-life traffic scenarios, near-optimal target states are hard to estimate. Moreover, it is noteworthy that the quantities of jerk at the endpoints of the uniform polynomials are arbitrary, depending on the position quintic polynomials which are determined by the trajectory boundaries in terms of distance, speed and acceleration. Consequently, unlike in the case of the multi-piece trajectories, the jerk continuity of consecutive plans in the context of uniform polynomials is hard to realize.

In general, the performance of the proposed trajectory generation strategy in terms of smoothness is promising, although there is still room for further improvement which is left for future work. With a high level of smoothness , the trajectories generated by the proposed planner can be more feasible than those produced by similar planners like the one illustrated in [23]. Some search-based motion planners like those presented in [25] [26] [27] can also obtain smooth trajectories by applying a post-optimization on a rough trajectory generated based on a lattice with fixed speed discretizations. The trajectories generated by them are less diversified than those applied in [23] and this work. Besides, rather than cubic polynomials applied in this work, their acceleration profiles are quadratic polynomials, which cannot guarantee jerk continuity at the joints of successive trajectory edges. The post-optimization can improve this situation, though. As mentioned in [25], the convergence of the post optimization relies largely on the quality of the initial guess, i.e, the rough trajectory generated based on the lattice. In this sense, the trajectories generated by the proposed planner can serve as high-quality initial guesses should a post-optimization be adopted. Implementing a lightweight post-optimization to further improve the smoothness of the trajectories should be considered

| Description | $k_{trans}$ | $\alpha_0$ $(m/s^2)$ | $v_0$ $(m/s)$ | $\alpha_1$ $(m/s^2)$ | $v_1$ $(m/s)$ | $\delta s$ $(m)$ | $\delta t$ $(t)$ | Jerk Level (multi-piece) | Jerk level (uni-form) | Visualization |
|---|---|---|---|---|---|---|---|---|---|---|
| Accelerate from stop | 0.5 | 0 | 0 | 1 | 14.14 | 100 | 14.39 | 2.4 | 0.273 | Figure 4.6(a) |
| Accelerate from stop | 2 | 0 | 0 | 1 | 14.14 | 100 | 15.13 | 0.6 | 0.175506 | Figure 4.6(b) |
| Decelerate and accelerate | 2 | 0.18 | 11.37 | 1 | 8.7 | 94 | 16.28 | 2.17 | 1.06 | Figure 4.6(c) |
| Stop from high speed | 1 | 0 | 30 | −4 | 0 | 284 | 16.9 | 4.77 | 1.59 | Figure 4.6(d) |

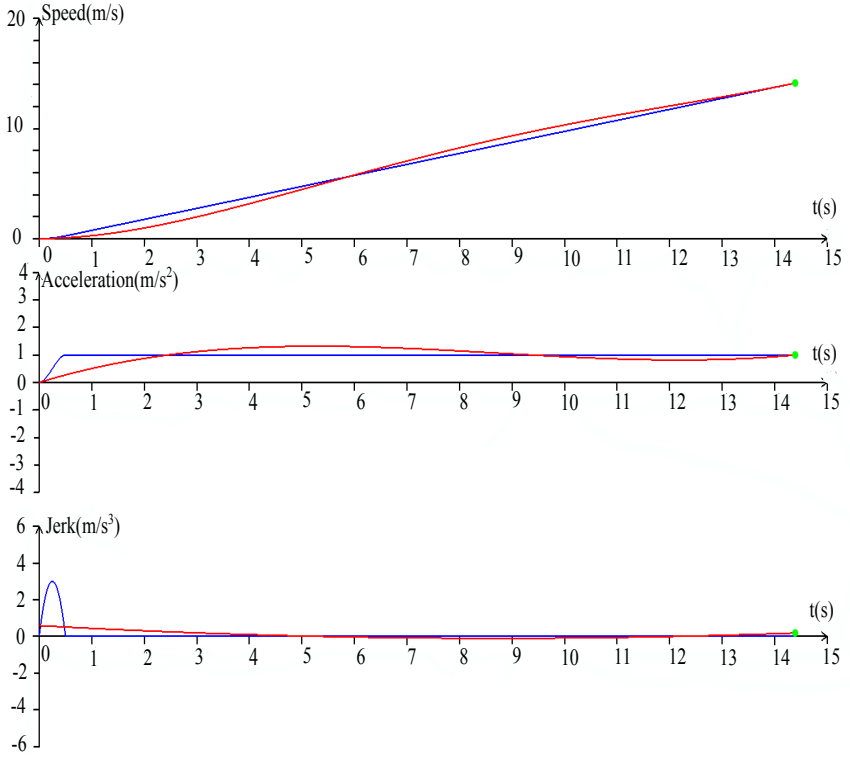
Table 4.2: Evaluation result of the smoothness of the trajectories generated based on the proposed trajectory representation strategy. The jerk level is calculated according to Equation 4.3. See Subsection 4.2.1 for the meaning of $k_{trans}$.

in future work. Another search-based motion planner proposed in [24] distinguishes itself from the motion planners discussed above as its path edges are not confined to a specific curvature polynomial(cf. Subsection 2.1.2). However, no details about the quality of the resultant trajectories and the corresponding computational efficiency are provided, which makes it impossible to compare it with the proposed planner.

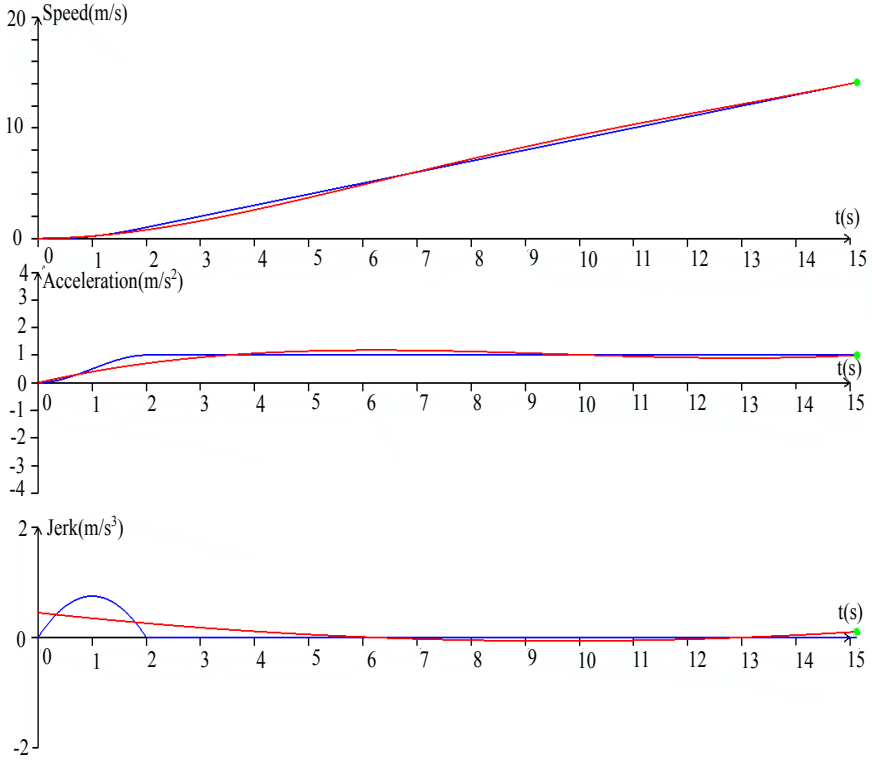
## 4.5 Summary

This chapter presented three kinds of acceleration profiles adopted in this work. They serve for the vehicle's smooth transition between different accelerations, its acceleration keeping and its reaching a specific state, respectively. These acceleration profiles are designed based on the theory of trajectories of minimum jerk. They are associated with the path edges to generate trajectory edges during the construction of the trajectories. The association approach applied in the proposed planner is helpful in improving the smoothness of the resultant trajectory from the perspective of a standalone planning horizon. Its main idea is to allow one acceleration profile to expand over several path edges, which distinguishes it from the approaches applied in similar works, where one acceleration profile can at most cover one edge.

In sum, the proposed trajectory representation strategy is distinguished in the smooth acceleration profiles and a method of associating multiple consecutive edges and one acceleration profile. This strategy can improve the feasibility of the trajectories generated by the planner.

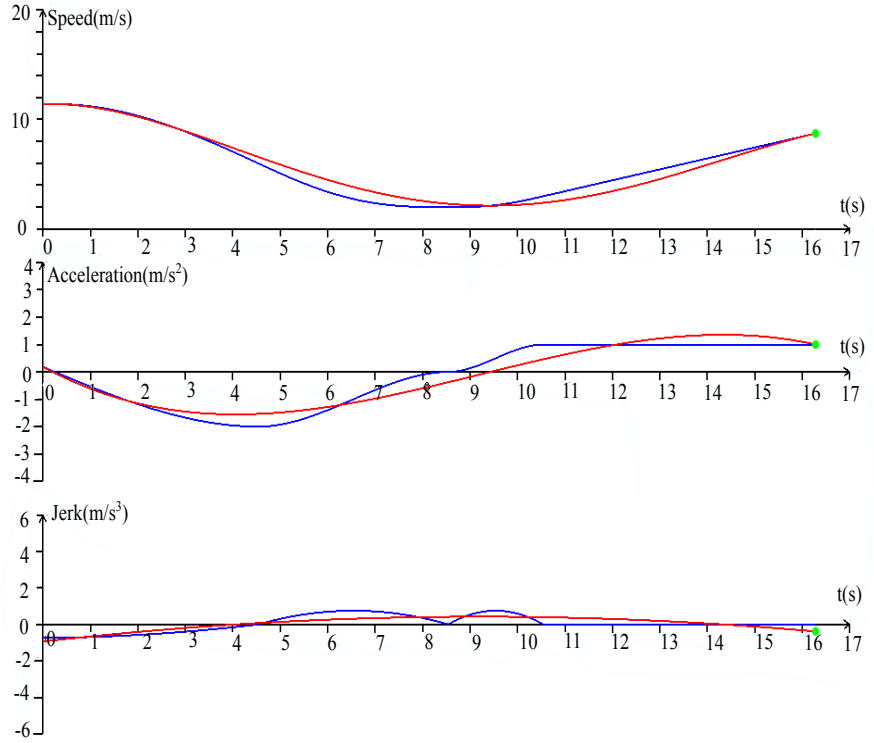



(a) Trajectories of speed, acceleration and jerk during an accelerating motion ($k_{trans} = 0.5$).

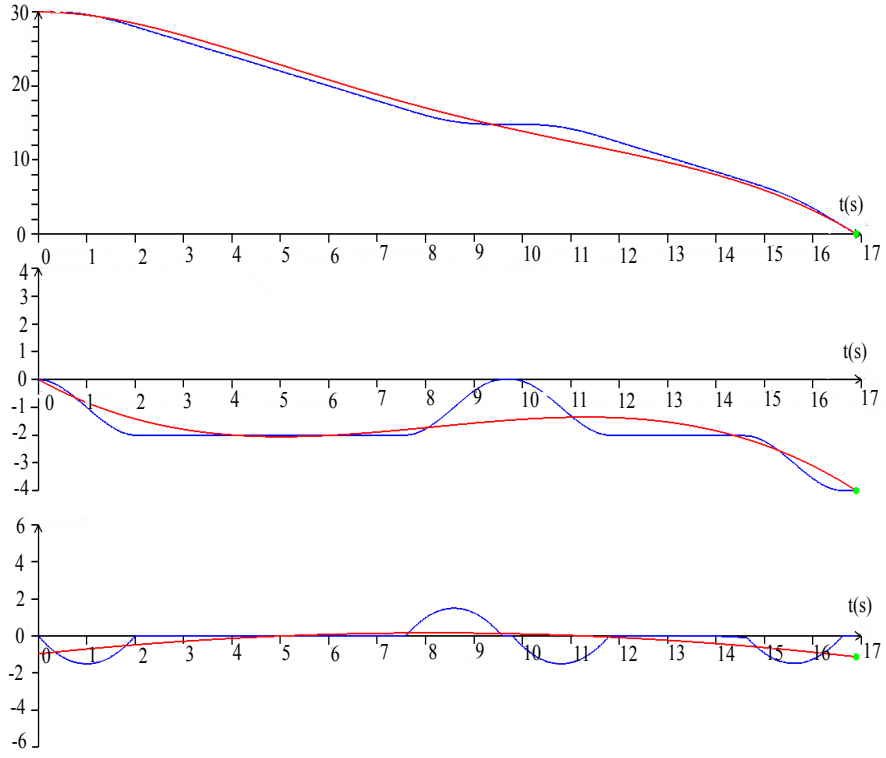



(b) Trajectories of speed, acceleration and jerk during an accelerating motion($k_{trans} = 2$).

Figure 4.6: Uniform polynomials and multi-piece trajectories of speed, acceleration and jerk during three typical vehicle motions. The red curves are the results of the uniform polynomial. The blue curves demonstrate the behaviour of the planner.




(c) Trajectories of speed, acceleration and jerk during a decelerating-accelerating motion ($k_{trans} = 2$).




(d) Trajectories of speed, acceleration and jerk during a decelerating motion ($k_{trans} = 1$).

Figure 4.6: Uniform polynomials and multi-piece trajectories of speed, acceleration and jerk during three typical vehicle motions. The red curves are the results of the uniform polynomials. The blue curves demonstrate the behaviour of the planner.

# Chapter 5

# Motion Planner Implementation and System Integration

This chapter describes the implementation of the proposed motion planner. Section 5.2 shows how the motion planning strategy and algorithms presented in the previous two chapters can be put into practice to generate an effective and practical motion planner. Besides, it lists the cost functions devised and implemented in the proposed planner. Section 5.3 begins with an introduction to the planning system of MIG and the interfaces between different planning modules. Then, the integration of the proposed planner into the planning system of MIG is demonstrated. Finally, the measures adopted to compensate for the planning latency and to promote the planning consistency between successive planning horizons are illustrated. Before that, we firstly take a look at how to define the planning horizon and the durations of the planning cycles.

## 5.1 Planning Horizons and Durations of Planning Cycles

At the start of each planning cycle, a planning horizon should be specified where the sampling can be performed. In the context of motion planning in the spatiotemporal space, the planning horizon is twofold: the temporal horizon embodied by a period of time and the spatial horizon consisting of road segments within a travel distance. Let the temporal horizon, the spatial horizon and the duration of one planning cycle be denoted as $H_T$, $H_S$ and $C_T$ respectively. $H_T$, $H_S$ and $C_T$ are among the most important design parameters of a motion planner. $H_S$ contains several aspects itself, such as its extending distance $d_H$, the number of the road segments $n_{seg}$ within its domain, the width of each road segment $w_{seg}$, etc. The strategy for specifying the road segments is discussed in the next section and is assumed given here. Consequently, among the aspects of $H_S$, only $d_H$ remains to be influenced; the others are decided by the method of specifying the road segments and the practically unchangeable road layout. As a result, the following focuses on how to specify $H_T$, $d_H$ and $C_T$.

In general, the values of design parameters should be decided based on the evaluation criteria and constraints of the design. Recall that the performance indexes of a motion planner include *completeness*, *feasibility*, *optimality* and *computational complexity* (cf. Subsection 1.3). The criterion *optimality* can be further divided into *horizon optimality*, *scenario optimality* and *resolution optimality* (cf. Subsection 1.3). Among those six criteria, horizon optimality and computational complexity are directly related to the choices of $H_T$, $H_S$ and $C_T$. Assuming that an intra-horizon planning strategy is given and that the vehicle's perception of the surrounding traffic is ideally sufficient, it is straightforward that an extensive planning horizon would improve the condition of horizon optimality. Due to limited computational resources, however, motion planning in an extensive planning horizon would inevitably require a long $C_T$, which is impractical due to safety concerns. The real-life traffic can be highly dynamic, and the vehicle's actual sensing ability is still limited. As a result, it is necessary for the motion planner to update its plans at a high frequency so that it can react quickly to newly perceived information. Considering all the factors mentioned above, the guidelines for choosing $H_T$, $d_H$ and $C_T$ are outlined as follows:

- From the perspective of horizon optimality, $C_T$ should be as large as possible. Conversely, the safety requirement imposes a limit on the upper bound of $C_T$. As a result, $C_T$ is determined based on the safety requirement. This constraint is obtainable from a comprehensive assessment of the vehicle's sensing and predicting ability and the complexity of the traffic environment.

- The lower bound of $H_T$ should be no less than $C_T$ as the vehicle must have a plan to follow while the motion planner is working on a new plan. In other words, it is required that the time domain of the trajectory generated from the last planning cycle should at least cover the duration of the current planning cycle. Given an average vehicle speed within the planning horizon, the lower bound of $d_H$ can be calculated according to the lower bound of $H_T$

- As the computational complexity can be expressed in terms of all the aspects of $H_S$, $d_H$ can be derived as an expression in terms of the computational complexity and the remaining aspects. Note that the maximum computational complexity in terms of time should not exceed $C_T$. Consequently, the upper bound of $d_H$ can be determined by a comprehensive effect of $C_T$ and the road layout aspects of $H_S$. Again, given an average vehicle speed within the planning horizon, the upper bound of $d_H$ can be used to set the upper bound of $H_T$. In this way, the value of $C_T$ and the ranges of $H_T$ and $d_H$ can be determined.

The concrete application of those principles listed above in determining $C_T$, $H_T$ and $d_H$ requires lots of experiments and repetitive adjustments, which is beyond the scope

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| Station increments | 10 | Station interval | $10\,m$ |
| Lattitude increments | 20 | Lateral offset | $0.5\,m$ |
| Outgoing paths of a single spatial node | 18 | Sampling unit of the XYSL Map | $0.5\,m \times 0.5\,m$ |
| Acceleration profiles | 8 | Spatial sampling unit of the cost maps | $0.5\,m \times 0.5\,m$ |
| Time discretizations | 2 | Path sampling step | $0.5\,m$ |
| Speed discretizations | 4 | Frames of the dynamic cost map | 20 |

Table 5.1: Parameters of the example state lattice

of this work. Accordingly, it is assumed in this thesis that reasonable values of $C_T$, $H_T$ and $d_H$ are given.

## 5.2 Motion Planner Implementation

The work flow of the proposed planner is demonstrated in Figure 5.1. The modules framed with dashed lines are carried out on the GPU, while the others are implemented on the CPU. Table 5.1 displays an example of the configuration of the lattice. Table 5.2 reports the time required to perform each phase of the planning cycle based on the lattice introduced in Table 5.1.

As can be seen from Table 5.1, the number of the outgoing paths of a single spatial node is relatively small in the example lattice. As a result, it takes only a small amount of time to construct the paths on the CPU, as is shown in Table 5.2. This is what happens in the current implementation of the proposed planner. If more paths were to be generated, it would be necessary to construct the paths on the GPU. Besides, it will also be beneficial to build the cost maps on the GPU.

In the rest of this section, the implementation of each phase of the planning cycle shown in Figure 5.1 is described in detail. The illustration starts with the construction of the spatial graph and cost maps that takes place on the CPU. Then, the path cost evaluation and the state lattice construction that are implemented on the GPU are presented. After that, the selection strategy of the best target node and the reconstruction and sampling of the best trajectory are described. Finally, the cost functions applied in the motion planner are listed.

### 5.2.1 Planner Implementation on the CPU

In the following, the phases of the planning cycle that are implemented on the CPU are described.

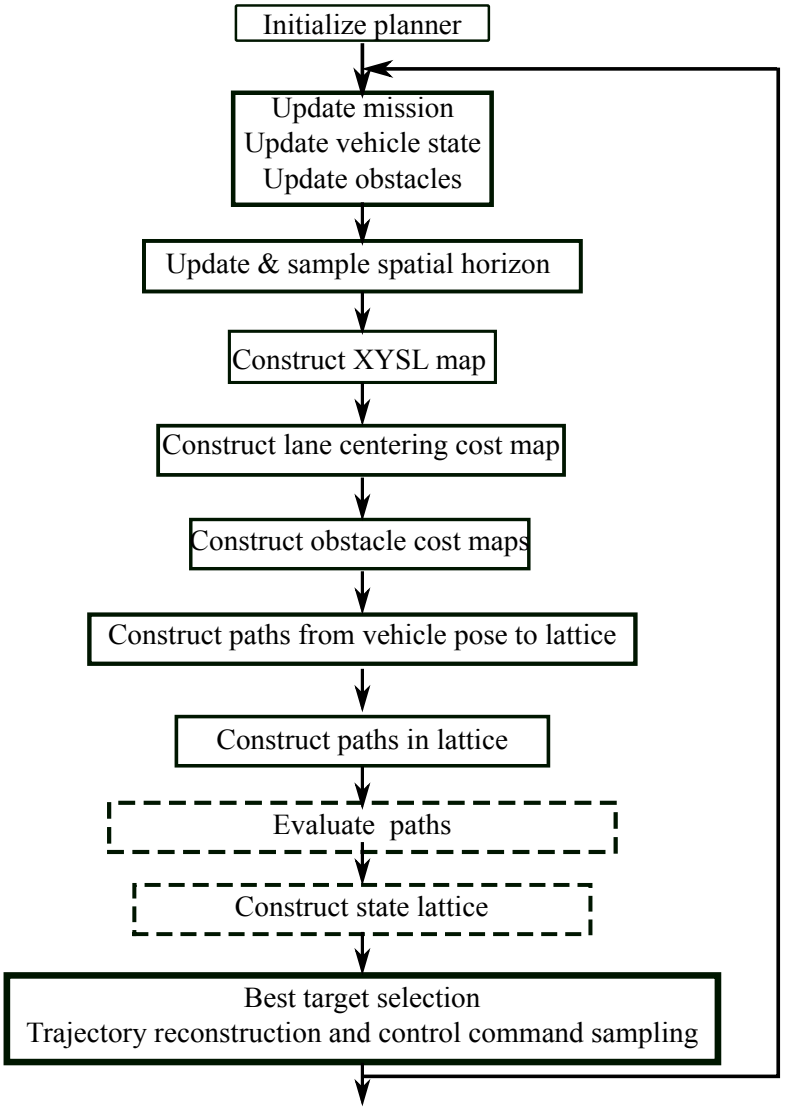



Figure 5.1: Work flow of the proposed planner. The modules framed with dashed lines are carried out on the GPU, and the others on the CPU.

#### 5.2.1.1 Planner Initialization

The initialization stage specifies the distance horizon, the horizon sampling units, the connectivity pattern, the types of acceleration profiles, the size and resolution of the time-speed grid, etc. These specifications are set once and subject to no alterations in the overall planning process. Although some parameters, such as the weights for the cost criteria and the sizes of the submaps of the cost maps, should be adjusted on the fly to adapt to different traffic scenarios and road conditions, they remain constant in the present implementation of the motion planner. Further improvements should be considered in future work. Besides, the precomputed initial guess table is also loaded into the memory at this stage. As is mentioned in Section 3.2.3, the initial guess for calculating the parameters of a path polynomial in runtime is obtained by querying the

| Stage | Time | Device |
|---|---|---|
| Spatial horizon specification and sampling | 0.8 $ms$ | CPU |
| Update of $XYSL$ map | 7 $ms$ | $CPU$ |
| Construction of lane centering cost map | 0.3 $ms$ | CPU |
| Update of static obstacle cost map | 0.3 $ms$ | CPU |
| Construction of dynamic obstacle cost map | 18 $ms$ | CPU |
| Generation of paths from vehicle to lattice | 0.05 $ms$ | CPU |
| Update of paths within lattice | 7 $ms$ | CPU |
| Data transfer from the CPU to the GPU | 1.5 $ms$ | CPU to GPU |
| Path cost evaluation | 0.018 $ms$ | GPU |
| State lattice construction | 189 $ms$ | GPU ( the number of evaluated trajectory edges $\approx$ 108869) |
| Data transfer from GPU to CPU | 1 $ms$ | GPU to CPU |
| Selection of the best target node and generation of control commands | 0.0007 $ms$ | CPU |

Table 5.2: Average time taken for each phase of the planning cycle.

| Parameter | Range | Sampling step |
|---|---|---|
| $\delta x = x_1 - x_0$ | $[1, 51]$ $(m)$ | 3 $m$ |
| $\delta y = y_1 - y_0$ | $[-10, 10]$ $(m)$ | 1 $m$ |
| $\delta\theta = \theta_1 - \theta_0$ | $[-90°, 90°]$ | $5°$ |
| $k_0, k_1$ | $[-0.2, 0.2]$ $(rad/m)$ | 0.04 $(rad/m)$ |

Table 5.3: Sampling scheme of the initial guess table.

initial guess table. The size and scale of the grid defining the initial guess table is shown in Table 5.3.

#### 5.2.1.2 Spatial Horizon Specification and Sampling

This procedure can be further divided into several sub-processes, as is illustrated in Figure 5.2. It would be ideal if the current vehicle state provided by the navigation system could be directly used for the construction of the spatial horizon. However, a planning cycle of $100 \sim 200\ ms$ causes a delay in the execution of the generated plan. Consequently, the vehicle start state for the planning should be a future state, i.e., ideally the state of the vehicle at the moment when the plan generated by the current planning cycle begins to be executed by the vehicle. In the proposed planner, such future state is predicted using a forward simulation of the vehicle dynamics based on the control commands that will be executed by the vehicle in the simulation duration. Besides, in order to guarantee a certain level of planning consistency, it is better to extend the

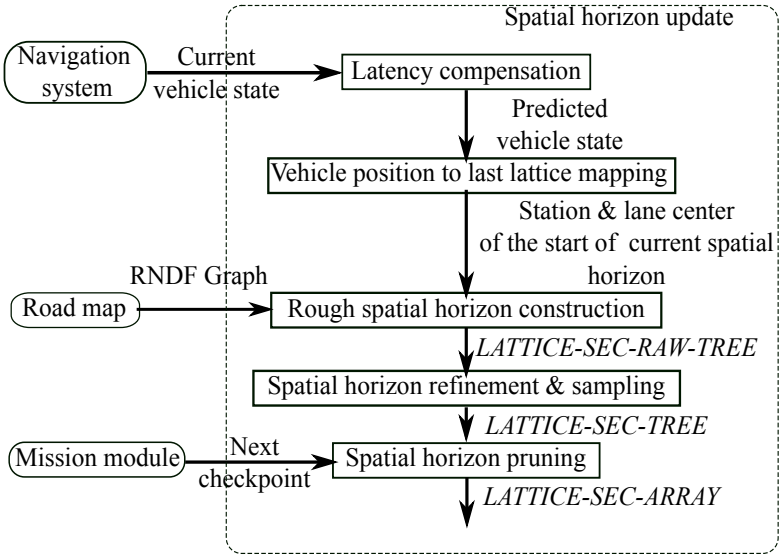



Figure 5.2: Update of the spatial horizon and spatial nodes.

spatial horizon based on the previous one, if it exists. This issue and those related to the compensation for the planning latency are discussed in detail in Section 5.3.3 and Section 5.3.2. Once the starting station of the planning horizon is determined, the spatial horizon gets constructed and the spatial nodes are sampled according to the algorithms and guidelines presented in Section 3.1 and Section 3.2. The output of this stage is a *LATTICE-SEC-ARRAY* (cf. Section 3.1) which records all the information of the spatial nodes for later use.

#### 5.2.1.3 Construction of the $XYSL$ Map

As is argued in Section 3.5, a discrete representation of the function that maps the $(x, y)$ coordinates to their corresponding $(s, l)$ coordinates is necessary for facilitating the construction of the cost maps. Following [23], this thesis regards this discrete representation as an $XYSL$ map. As the spatial horizon defined in the proposed motion planner might contain several road segments, at least one submap of the $XYSL$ map is defined for each segment. The $(s, l)$ coordinates corresponding to the point $P(x, y)$ that represents a cell of the $XYSL$ map are calculated by implementing a recursive search on the segment of the center line bounded by the $XYSL$ map. The search is intended to find the point $p_c$ on the center line that is closest to the point $P(x, y)$ in question ( denoted as $p_0$ ). With $p_c$ located, the distance between $p_c$ and $p_0$ serves as the lateral offset, i.e., the absolute value of the $l$ coordinate of $p_0$. The relative position of the two points and its relationship with the direction of the tangent line of the lane center at $p_c$ are exploited to determine the sign of the $l$ coordinate. The $s$ coordinate is equal to the arc length of the segment of the center line between the starting station of the planning horizon and the station of $p_c$ which can be readily extracted from the representation of the center line.

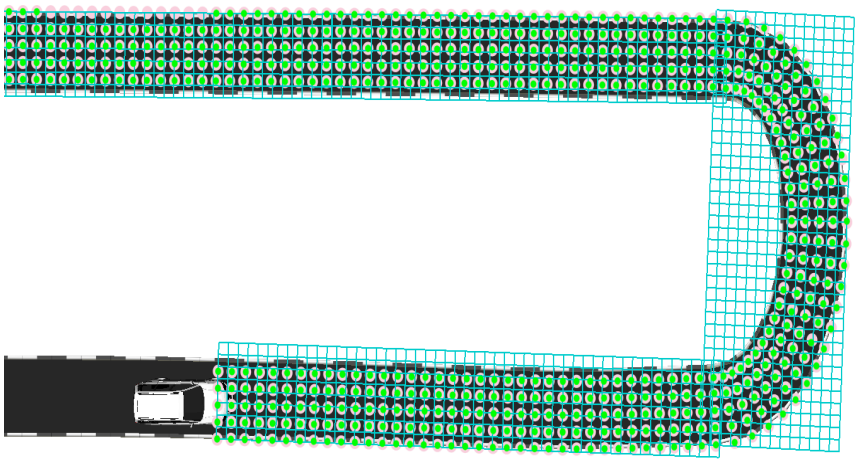


Figure 5.3: Mapping results $(s, l)$ (pink points) of points $(x, y)$ (green points) based on the $XYSL$ map defined on the grids shown in cyan. The size of each grid cell is $0.5\,m \times 0.5\,m$.

Given the $XYSL$ map, the $(s, l)$ coordinates of an arbitrary point $P(x, y)$ in the spatial horizon are obtained by following the procedures listed below:

- Locate the submap whose domain (in the sense of the continuous space) contains $P(x, y)$.
- Get the four grid cells whose representative points lie nearest to $P(x, y)$.
- The $(s, l)$ coordinates are calculated by a bilinear interpolation on the four neighbouring grid cells.

Figure 5.3 demonstrates the accuracy of such mapping.

#### 5.2.1.4 Update of the Lane Centering Cost Map

The domain of the lane centering cost map is the same as that of the $XYSL$ map. The cost of the representative point $(x, y)$ of each cell of the cost map is evaluated using the following functions:

$$\begin{aligned} cost(x, y) &= cost(XYSL(x, y)) = cost(s, l) \\ cost(s, l) &= \begin{cases} \infty & \text{if } l \in \delta l_{curb} \\ c_{on-coming} + k_{LC}|l - l_{cl_{on-coming}}| & \text{if } l \in \delta l_{on-coming} \\ k_{LC}|(l - l_{cl})| & \text{otherwise} \end{cases} \end{aligned} \tag{5.1}$$

where $k_{LC}$ is a constant slope, making the cost increase linearly as the deviation between the trajectory and the center line grows, $c_{on-coming}$ is the penalty for the vehicle's running on the lanes with oncoming traffic, $l_{cl}$ is the $l$ coordinate of the center line,

$\delta l_{curb}$ and $\delta l_{on-coming}$ refer to the latitude ranges of the road curbs and the roads with oncoming traffic respectively.

Following the $XYSL$ map, the lane centering cost map may contain $M$ submaps, where $M > 1$. Each submap is represented as a set $\{C(i,j), i = 0,1,...,N_x - 1, j = 0,1,...,N_y - 1\}$ and stored in a subarray $\{\hat{C}_k, \hat{C}_k = C(i,j), k = j + N_y i\}$. The subarrays storing the submaps are put together to compose the complete array for the cost map. The starting index of the $m$th subarray is equal to the summed size of the $(m-1)$ subarrays before it. As the meta information of the submaps, such as the reference frames and sizes of the submaps, are necessary for the lookup of costs, they are also recorded in separate arrays, each having a size of $M$. Figure 5.4 demonstrates an example cost map that is stored in this way. Such storage scheme will facilitate the uses of the cost maps on the GPU.

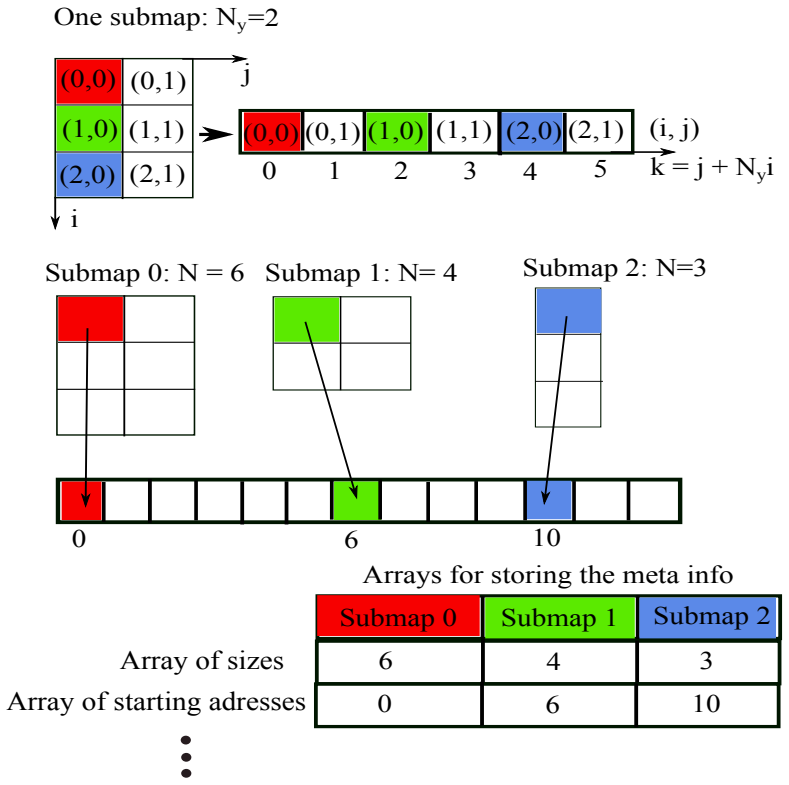



Figure 5.4: Storage strategy of the lane centering cost map.

#### 5.2.1.5 Construction of the Obstacle Cost Maps

In the perception system, the bounding boxes of the detected obstacles are organized in a spatial *kd-tree*(*Skd-tree*)(cf. [84]), where each bounding box serves as a leaf node of the tree. This tree structure is adopted for the purpose of making collision checking efficient. A point is regarded as being trapped in collision if it is contained in one of the leaf nodes of the *Skd-tree*. Each cell of the spatial grids over which the cost map are defined is subject to such a collision checking. That is, if a cell hits one of the leaf nodes

of the *Skd-tree*, it will be assigned a value of $\infty$ which means that it is untraversable. In this way, the original cost map of static obstacles and each frame of the original cost map of dynamic obstacles are constructed. Here the original cost maps refer to the cost maps without dilation. The dilation follows the rules and procedures described in Section 3.5.2. Finally, the dilated cost maps are stored in a way similar to the lane centering cost map.

#### 5.2.1.6 Storage of the Path Edges

The information pertaining to a path edge consists of several components, such as the five parameters defining the cubic spiral that represents the path edge, as well as the target node of the path edge. In this sense, the information of all the path edges can be organized in the form of an array of structures, as demonstrated in Figure 5.5. For reasons discussed in the next subsection, an array of structures should be transformed into a structure of arrays, as shown in Figure 5.5. A path edge is indexed $(i_s, i_l, i_c)$, where $(i_s, i_l)$ is the index of the source node of the path edge, and $i_c$ is the index of the path edge in the group of path edges outgoing from the same source node in accord with the connectivity pattern. The elements of the arrays where the information of the path edge $(i_s, i_l, i_c)$ is stored are indexed by $i_s N_l N_c + i_c N_l + i_l$, where $N_l$ is the number of lateral increments in the lattice, and $N_c$ the number of path edges outgoing from a single spatial node. The reason behind this indexing scheme can be found in Subsection 5.2.2. As all the path edges from the vehicle to the lattice have the same source node (i.e., the vehicle start pose) and have a different connectivity pattern from those within the lattice, the two kinds of path edges are stored separately.

Array of structures

| Index | Path polynomial |
|---|---|
| 0 | $P_0(a_0, a_1, a_2, a_3, s_T)$ |
| 1 | $P_1(a_0, a_1, a_2, a_3, s_T)$ |
| 2 | $P_2(a_0, a_1, a_2, a_3, s_T)$ |

Structure of arrays

| Index | array-$a_0$ | array-$a_1$ | array-$a_2$ | array-$a_3$ | array-$s_T$ |
|---|---|---|---|---|---|
| 0 | $P_0.a_0$ | $P_0.a_1$ | $P_0.a_2$ | $P_0.a_3$ | $P_0.s_T$ |
| 1 | $P_1.a_0$ | $P_1.a_1$ | $P_1.a_2$ | $P_1.a_3$ | $P_1.s_T$ |
| 2 | $P_2.a_0$ | $P_2.a_1$ | $P_2.a_2$ | $P_2.a_3$ | $P_2.s_T$ |

Figure 5.5: Storage of path edges.

### 5.2.2 Planner Implementation on the GPU

The most expensive procedure in the proposed motion planner is the construction of the state lattice during which tens of thousands of trajectories are generated and evaluated. As is discussed in Section 3.4, the construction can also be regarded as a search process

from the perspective that trajectories with less desirability are abandoned during the construction before they reach the end of the planning horizon. The parallel computing potential of GPUs is explored in the proposed planner to handle the intensive computation. In the following, the GPU-based implementation of the graph construction algorithm is described. Before that, the evaluation of the path edges in terms of static cost which is also implemented on the GPU is illustrated. To begin with, an introduction to the basic concepts of the GPU computing architecture is provided, which is helpful in understanding the presented GPU-based application.

#### 5.2.2.1 Parallel Computing Architecture of GPUs

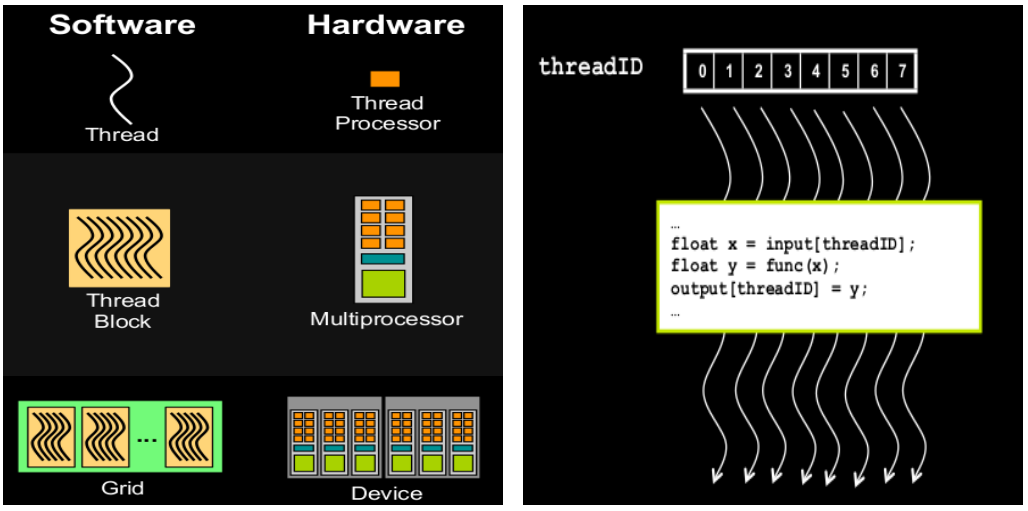
Compute Unified Device Architecture (CUDA) is a parallel programming model and software environment for parallel computing supported by most of recent Nvidia GPUs. The basic conceptual elements of CUDA and their counterparts on the GPU hardware (as shown in Figure 5.6(a)) are the following:

- A thread which is executed by a thread processor.
- A thread block containing many threads which is executed in a multiprocessor. The threads are further divided into groups of 32 parallel threads called warps. The multiprocessor works by creating and managing these warps.
- A grid of thread blocks that is executed on a device with several multiprocessors.

The *kernel* is the function that is executed by an array of threads simultaneously, as is shown in Figure 5.6(b). Each thread has a unique ID which can be used by the kernel to get thread-specific memory addresses. Figure 5.7 illustrates how the kernel accesses the local and global memory on the GPU device and how data are transferred between GPUs and CPUs. In essence, the parallel computing on GPUs is realized by launching batches of threads that execute the same kernel, while the processed data and conditional instructions designated by the thread ID might be different among the threads.

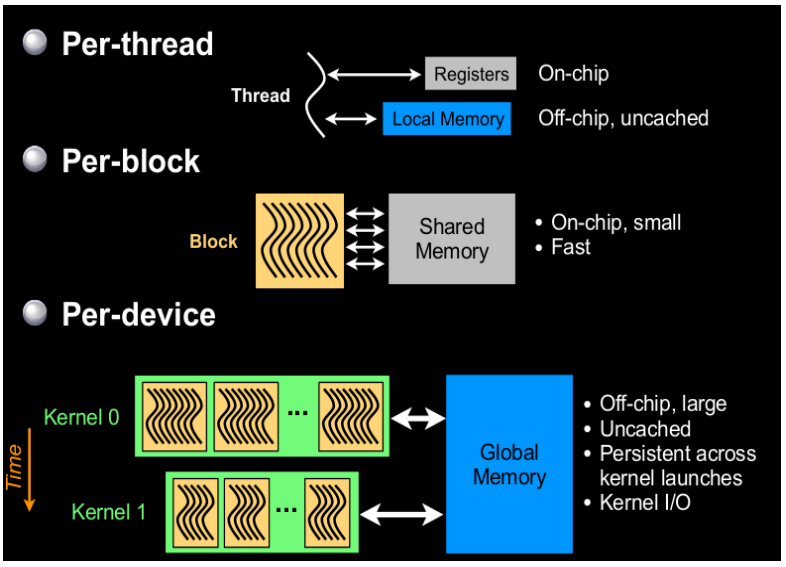
Some optimization strategies are suggested in [85] which are helpful in achieving maximum performance of the parallel computing architecture. They are concerned with mainly three aspects, i.e., maximization of execution parallelism, optimization of memory usage and maximization of instruction throughput. The second point is taken into account in the presented implementation, while the first and the third ones are not touched in this thesis and should be addressed in future work.

The first step to maximize memory throughput is to avoid as much as possible data transfers with low bandwidth and high latency. Referring again to Figure 5.7, the memory transfer between GPUs and CPUs has the lowest bandwidth and the highest

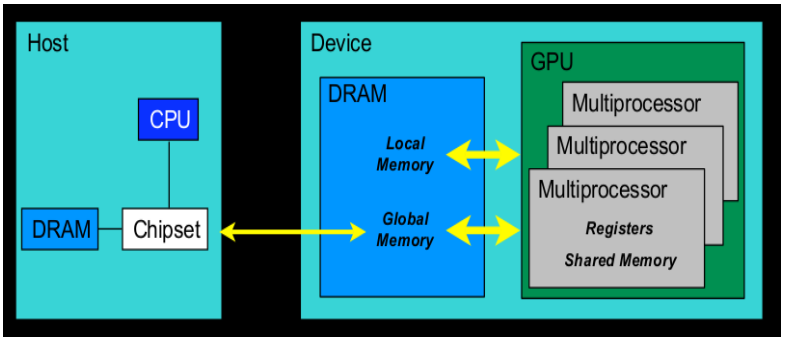



(a) Execution model of CUDA. (b) A kernel executed by an array of threads.

Figure 5.6: Basic concepts of GPU computing model, from [85]




(a) Memory accesses on the GPU device.




(b) Data transfers between CPUs and GPUs.

Figure 5.7: GPU-based Memory handling, from [85]

latency. Memory accesses of global memory are also inefficient, but better than GPU-CPU memory transfers. Compared to them, shared memory and caches have much higher bandwidth and much lower latency.

The second step is to organize memory accesses as optimally as possible, no matter what kind of memory access is used. Most memory accesses in the presented application are in the form of global memory requests. Consequently, optimal patterns for global memory accesses should be applied in the implementation as much as possible. Global memory is accessed via 32-, 64-, or 128-byte memory transactions. These memory transactions can only be executed for aligned memory segments of 32-, 64-, or 128-byte whose first address is a multiple of their size. Global memory accesses of the threads within the warp are coalesced into one or more of these memory transactions. The fewer transactions are necessary, the fewer unnecessary memory units are accessed. As a result, maximizing coalescing is crucial in maximizing the global memory throughput. The rules for coalescing vary with the compute capability of the GPU. For GPUs with compute capability 1.0 and 1.1, consecutive threads of a half-warp must access consecutive memory units and the memory accesses of a half-warp can only be coalesced when they request a single aligned memory segment. These constraints are much more relaxed for GPUs with compute capability 1.2 and 1.3, where memory requests from non-neighbouring threads can be coalesced as long as their targets can form a single aligned memory segment. For GPUs with compute capability 2.x and higher, the memory transactions are cached, so in addition to coalescing, it is also necessary to exploit data locality in order to maximize the memory throughput.

For the current GPU-based implementation of the proposed planner, a certain amount of data need to be transferred from the CPU to the GPU, such as the information of the path edges and the cost maps. Besides, after the construction of the state lattice is finished, it is necessary to transfer the information of the constructed lattice nodes back to the CPU to be further processed there. As far as the current application is concerned, the cost of the transfer is limited and tolerable (cf. Table 5.2).

The GPU used in this work has compute capability 2.0. Although the constraints on coalescing for such GPUs are not very strict, it is still beneficial to follow the suggestion that consecutive threads access consecutive memory units where it is possible and convenient to do so.

#### 5.2.2.2 Path Sampling

Before the state lattice is constructed, the static costs of the path edges should be evaluated. The reason why the paths are not evaluated during the construction of the state lattice is twofold. First, unlike the trajectory edges that have to be constructed during the search process, the path edges are fixed as long as the spatial nodes and the connectivity pattern are given. As a result, it is possible to evaluate the path edges

before the construction of the graph. Second, should the evaluation take place during the graph construction process, one path edge might be evaluated many times, as several trajectory edges may be generated based on the same path edge. Accordingly, evaluating the paths during the graph construction process will hamper the computational efficiency of the planner.

The path edges are sampled at even intervals in terms of arc length by using the trapezoidal integration (cf. Section 3.2.3). Each sample $(x, y, \theta, \kappa)$ is evaluated for the static cost. Each path is sampled and evaluated in a separate thread executed on the GPU, as is displayed in Figure 5.8.

In [23], the samples of the paths are also employed for the dynamic cost evaluation of the trajectories. It is viable to do so in the planner proposed in [23], because only constant accelerations are applied in [23] which makes it easy to calculate the time and speed of one sample given the time and speed of the preceding sample. However, from the perspective of the proposed planner where acceleration cubic polynomials are employed, evaluating the dynamic cost based on the existing path samples means solving for the roots of position quintic polynomial equations, which is hard to implement. As a result, the path is sampled again for the dynamic cost evaluation. Both samplings follow the trapezoidal integration formula. The difference lies in that the samples for the dynamic cost evaluation are spaced at even intervals in terms of time rather than arc length as in the sampling of paths for the static cost evaluation. Section 4.3 gives more details on the dynamic cost evaluation of the trajectories.

#### 5.2.2.3 State Lattice Construction

As is mentioned earlier, the path edges are multiplied into several trajectory edges by associating acceleration profiles with them. Trajectory edges that arrive at the same lattice node $N(i_s, i_l, i_\alpha, i_v, i_t)$ are subject to a pruning operation. The speed and time of the end state of the trajectory edge that survives the pruning operation are selected to represent the lattice node. The traversal cost of the trajectory segment from the vehicle to this newly represented node is recorded in a cost map. Each cell of this map is indexed by $(i_s, i_l, i_\alpha, i_v, i_t)$ and corresponds to the lattice node in the state lattice indexed in the same way. Besides the traversal cost, the time and the speed, the information recorded in the cell also contains:

- The index of the parent node of the current node, i.e., the source node of the trajectory edge $e_c$ that ends at the current node.
- The coefficients, starting time, starting speed and duration of the last acceleration cubic polynomial along the current trajectory edge. Note that *the last polynomial* is stressed here because a trajectory edge might contain one or two acceleration cubic polynomials. Referring to Chapter 4, if there happen to be two polynomials,

the first one is the remaining segment of the acceleration profile applied on the preceding trajectory edges. In this case, the information of the first polynomial must already be recorded at the preceding node. As a result, it is no longer necessary to record it again, unless it is still unfinished within the current edge $e_c$. The information of the polynomial is necessary for the construction of the subsequent trajectory edges. Besides, it is also indispensable for the reconstruction of the best constraint-abiding trajectory after the state lattice construction is finished and the best target node is selected.

- The time index of the first sample on the remaining segment of the last profile on $e_c$ for which the dynamic cost will be evaluated on the subsequent edges. This is only useful when the profile is unfinished within the current trajectory edge. In this case, the dynamic cost evaluation of the subsequent trajectory edges will start at this sample.

- The sampling interval in terms of time of the last profile on $e_c$ for dynamic cost evaluation. It is calculated as

$$\frac{\text{PROFILE-DURATION}}{\frac{\text{PROFILE-ARC-LENGTH}}{\text{PATH-SAMPLING-STEP}} + 1}. \tag{5.2}$$

  When the profile is unfinished on $e_c$, its remaining part will be evaluated with the same sampling interval on the next edge.

- The curvature of the last sample on $e_c$. This curvature, in combination with the curvature of the first sample on the following edge, is used to calculate the rate of change of curvature which is one subject for the dynamic cost evaluation.

- The arc length of the unfinished segment of the current acceleration profile. This is necessary for the construction of the subsequent trajectory edge.

- The desirability of the target node $n_1$ of $e_c$ in terms of speed and time. The sum of this cost term and the traversal cost will be used for the pruning operation to select the best candidate for representing the state lattice node that $n_1$ coincides with. Should $n_1$ survive the pruning, both the aggregate cost and the desirability cost will be stored at the corresponding cell of the cost map at the end of the current thread. At the beginning of the thread where the subsequent trajectories starting from $n_1$ are constructed, the desirability cost will be subtracted from the aggregate cost, which results in the traversal cost of the trajectory from the vehicle start state to the current node. This traversal cost will be used for the cost evaluations of the subsequent edges.

The cost map is stored in several arrays. Each array records one piece of the information listed above. The array element corresponding to the grid cell $(i_s, i_l, i_\alpha, i_v, i_t)$ of the cost map is indexed by:

$$idx_N = i_s n_\alpha n_v n_t n_l + i_\alpha n_v n_t n_l + i_v n_t n_l + i_t n_l + i_l$$

where $n_\alpha, n_v, n_t, n_l$ refer to the number of acceleration profiles, the number of speed discretizations, the number of time discretizations and the number of latitude discretizations in the $SL$ frame. Recall that the state lattice may contain several sections. Accordingly, the overall size of the state lattice is given as:

$$sum_{idx}(n_{sec}) = \sum_{i_{sec}=0}^{n_{sec}-1} n_s(i_{sec}) n_\alpha n_v n_t n_l(i_{sec})$$

where $n_{sec}$ is the number of sections, $n_s(i_{sec})$ and $n_l(i_{sec})$ refer to the number of the station discretizations and the number of latitude discretizations in the $SL$ frame of the $i$th section. In this way, the indices of the lattice nodes of one section begin with:

$$idx_{start}(i_{sec}) = \begin{cases} 0 & \text{if } i_{sec} = 0 \\ sum_{idx}(i_{sec}) & \text{otherwise} \end{cases}$$

These starting indices of the sections are also stored in an array to facilitate the calculation of the index of the lattice nodes in the cost map.

Figure 5.8 demonstrates how the threads for constructing the trajectory edges interact with the storage of the cost map and path edges. The construction of the trajectories is conducted in order of station. Each state lattice node initiates a thread executing the kernel which implements the algorithm demonstrated in Algorithm 1. In general, at most $N_c N_\alpha$ trajectory edges can be constructed in one thread which result in $N_c N_\alpha$ candidate nodes. Note that $N_c$ refers to the number of the path edges outgoing from one lattice node according to the definition of the connectivity pattern. It might be intuitive to make the kernel write the information of the candidate node $n_c$ to the corresponding memory of the to-be-represented node $n_s$ if the cost of $n_c$ turns out to be smaller than the cost of the current representative state of $n_s$. Recall that there might be several threads executing the instructions of the kernel concurrently. Consequently, it might happen that some of the threads need to modify the same memory segment as the trajectories generated by them end at the same to-be-represented node. To avoid race conditions, the built-in atomic function *atomicMin()* is employed to assign the smallest cost to the memory segment. *atomicMin()* ensures that the memory address can only be accessed by a second thread after the first thread completes its operation on this

memory address. If several candidate nodes turn out to have the same cost which happens to be the smallest cost, it is defined that the one generated by the thread with the smallest thread index will win, as is shown by the second *atomicMin()* in Figure 5.8. This strategy is just used for the purpose of convenience; there is no reason behind it (since the decision of the pruning selection is based solely on the cost, the candidate nodes with the same cost are supposed to be treated equally).

The implementation of the pruning process illustrated above works if all the threads that are initiated by the lattice nodes of a single station have strict step-wise synchronization. Otherwise, the construction of the state lattice will become crazy. For example, the thread that generates a candidate trajectory whose ending node does not have the minimum cost might be able to pass the tests of the two *atomicMin()* and writes the information of its ending node to corresponding memory, whereas the thread that has the minimum-cost candidate node might be kept back. As a result, it is necessary to check how the threads are synchronized in the computing architecture of CUDA. It turns out that only the threads that belong to the same block ( cf. Figure 5.6 for the concepts of block and thread) are strictly synchronized on current CUDA-enabled GPUs. That is, the threads that belong to different blocks may not execute the same instruction at the same time. Consequently, the implementation of the selection of the best candidate node presented above should be modified. The common method used to realize a global synchronization is to invoke a kernel from the host, as it is regulated that only after all the threads executing the current kernel are finished can the new kernel be launched. Accordingly, it is common practice to split a kernel at the points where a global synchronization is required and launch the resultant sub-kernels sequentially. The current execution of the kernel for constructing the trajectory edges and selecting the best candidate node adopts that "pipeline" method, as is shown in Figure 5.9.

As can be seen in Figure 5.8 and Figure 5.9, the storage schemes of the path edges and the lattice nodes can facilitate contiguous memory accesses among consecutive threads, which is helpful in maximizing the memory throughput of CUDA-based applications.

The algorithm for the application of the acceleration profiles presented in Section 4.3 is implemented in the threads for the construction of the trajectory edges demonstrated in Figure 5.8. Recall that extra nodes will be generated to record the stopping states of the vehicle that occur somewhere between connected lattice nodes should there be any. Such extra nodes are called *EXTRA-SAMPLE*s (cf. Section 4.2). The indices $i_v$ and $i_t$ of the cell of the cost map where the information of an *EXTRA-SAMPLE* is stored are zero and the maximum time index, respectively. Its indices $i_s, i_l$ and $i_\alpha$ are set in accord with the intended target node of the trajectory along which the *EXTRA-SAMPLE* in question occurs. In addition, the representative time value of the cell is set to $\infty$, communicating a message to the subsequent threads initiated by this particular cell that no further expansion should be carried out after it.

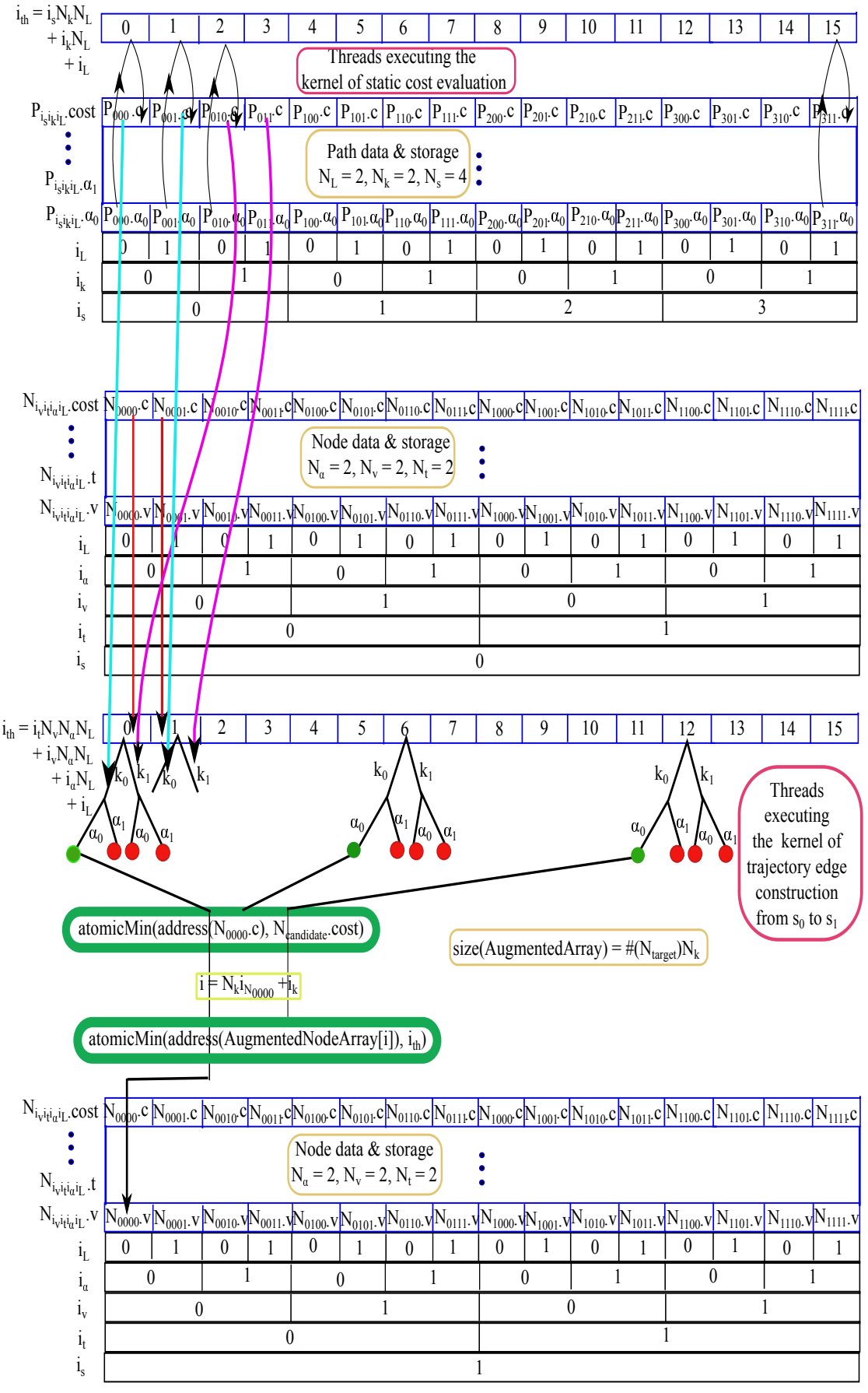



Figure 5.8: Static cost evaluation of path edges and the construction of the trajectories outgoing from a single station on the GPU.

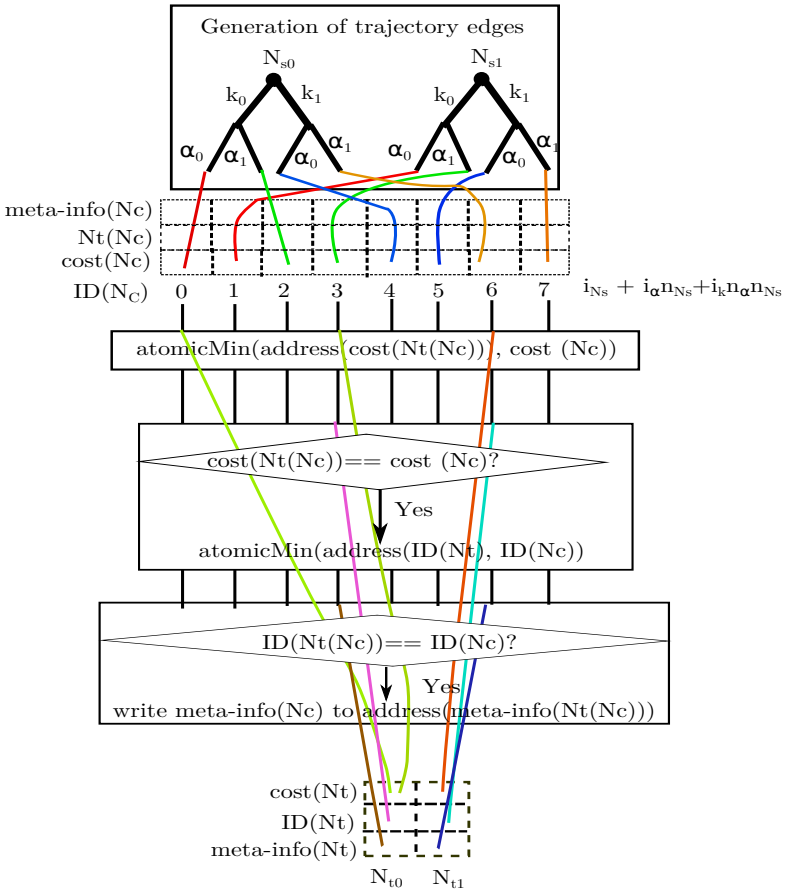



Figure 5.9: The kernel pipeline for constructing the trajectory edges and selecting the best candidate node.

### 5.2.3 Best Target Selection and Trajectory Reconstruction

After the construction of the state lattice is finished, the cost map that records the information of the lattice nodes are copied back to the CPU. On the CPU, the selection of the best target node and the reconstruction of the best constraint-abiding trajectory take place. The best target is selected among the nodes that belong to one of the five categories:

- Lattice nodes with maximum time index,
- Lattice nodes with maximum station index.
- *EXTRA-SAMPLE*s.
- Nodes with their representative speeds being zero. This kind of nodes will be further evaluated based on the assumption that the stopping state continues until the maximum time is reached.
- The vehicle start node. This node is only taken into account if its speed is zero.

In sum, the selected nodes should be able to reach the boundary of the planning horizon, in terms of either time or travel distance. This strategy is applied as it is believed that the nodes (with those belonging to the five categories listed above being excluded) that fall inside the planning horizon are either intermediate nodes along trajectories whose ending nodes reach the boundary of the planning horizon or nodes leading to no traversable trajectories. It is hard to devise criteria to evaluate the former against the nodes at the boundary that they will eventually lead to, while it is not necessary to check the latter as they are invalid. As a result, only the nodes satisfying the five categories have accesses to the selection.

The selection criteria take into account the following aspects:

- Traversal cost recorded in the node.
- Whether the representative time of the node is smaller than the planning cycle. If yes, the node should be punished severely.
- The cost of the target node in terms of its representative time and its station. For the purpose of promoting time efficiency, the larger the representative time is, the larger the additional cost is imposed on the target node. In order to encourage the trajectories to cover the whole planning horizon, the nodes with the maximum station index are given a bonus.
- The cost in terms of the difference between the speed at the target node and the target speed required by specific scenarios. The nodes whose representative speeds cannot guarantee a smooth transition to the target speed will get penalized.

After the best node comes out, the trajectory from the vehicle to the best node is reconstructed. As the lattice nodes record the nodes preceding them, all the nodes along the trajectory in question can be readily obtained. With these nodes at hand, the related path edges and acceleration profiles can be extracted with ease. Finally, the trajectory can be restored by associating the acceleration profiles with the path edges in a way similar to the way the trajectory is constructed from scratch. The control commands are also sampled during the process of the reconstruction of the best trajectory.

### 5.2.4 Cost Functions

In [23], a quite thorough discussion about the design principles and concrete applications of cost functions is presented for a motion planner with performance expectation and practical problems similar to the proposed planner in this thesis. To avoid redundancy, the cost functions applied in this work are listed in Table 5.4 and Table 5.5 only to provide a general overview. The reader is referred to [23] for more detailed explanation. It should be pointed out that the design of the cost functions and tuning of their parameters require repetitive adjustments according to the feedback of the planner's performance

in various experiments. Automating this progressive and painstaking learning process using techniques such as machine learning is one subject of future work.

## 5.3 System Integration

The proposed planner is integrated into the existing planning system of MIG for evaluation. As is shown in Figure 5.10, the core planner of the existing system is a path planner. Based on the world model and the vehicle state provided by the perception system, the path planner generates a path spline and a series of traffic control devices (e.g.,traffic light, stop sign) that the vehicle will encounter along the path spline (cf. [86]). The path spline generated by the path planner, together with the current position and driving direction of the vehicle, is used by a *pure pursuit* path tracker (cf. [16][87]) to calculate the steering commands for the vehicle to follow. The traffic control devices play an important role in the determination of the desired speed of the vehicle. The difference between the desired and current speeds of the vehicle is fed into a proportional-integral-derivative (PID) controller to generate the desired control signal for either the throttle or the brake actuator of the vehicle. This control signal, after being output from the tracker, is still subject to the verification by the safety layer, i.e, the module of reactive braking. Finally, the "risk-proof" actuation commands are sent to the actuators via the car gate module.

The planning architecture based on the proposed planner is shown on the right of Figure 5.10. It should be pointed out that the embedding of the scenario reasoning module is out of the scope of this work. Specific measures will be taken to "mimic" the scenario reasoning module for the evaluation of the proposed planner. Besides, a controller that can calculate the actuation commands based on the nominal trajectory issued by the motion planner is also necessary but absent from the current planning system. As designing a controller capable of handling complicated vehicle dynamics at high speed is also beyond the scope of this thesis, only a controller that is necessary for the evaluation of the planner in the simulation environment is constructed in this work. This controller is designed based on the nonholonomic vehicle model applied in the simulator. The limitations of such a controller will be further discussed in Chapter 8.

In the rest of this section, the work flow of the planning system based on the proposed planner is first described in detail. After that, issues related to planning consistency are discussed.

### 5.3.1 Planning Architecture based on the Proposed Planner

As mentioned earlier, by the time the trajectory is generated from the current vehicle state, the vehicle has moved on. A forward simulation of the vehicle system dynamics is therefore necessary for compensating for the planning latency. Given the estimated future state of the vehicle and the related road model, the spatial horizon can be specified.

| General purpose | Cost function | Description |
|---|---|---|
| Trajectory consistency between successive planning cycles | $c_{path}^{follow} = \begin{cases} -bonus & \text{if last path is followed} \\ 0 & \text{otherwise} \end{cases}$ | Bonus is bestowed to the candidate trajectory whose first two spatial nodes can be found in the last plan. |
| | $c_{\alpha}^{follow} = \begin{cases} -bonus & \text{if last pofile is continued} \\ 0 & \text{otherwise} \end{cases}$ | As the initial acceleration of the vehicle in the current planning cycle is extracted from the last plan (cf. Section 5.3.2), it might fall between the endpoints of an acceleration profile. In this case, the trajectory can get a bonus if its first acceleration profile is intended to finish that profile. |
| Trajectory stability within one planning horizon | $c_{lateral}^{variation} = \sum_{i=1}^{n}(l_i - l_{i-1})$ where $l_i$ is the latitude of the $i$th node along the trajectory | The variations in the latitudes of the spatial nodes along the trajectory result in penalty. |
| | $c_{acceleration}^{variation} = \begin{cases} -bonus & \text{if } \alpha_i \equiv \alpha_{i-1} \\ 0 & \text{otherwise} \end{cases}$ where $\alpha_i$ is the index of the $i$th acceleration profile along the trajectory. | The variations in acceleration profiles along the trajectory lose bonus. |
| Safety | $c_{static}^{obstacle}(x,y) = \begin{cases} \infty & \text{if (x,y) is in collision area} \\ \text{HIGH-COST} & \text{if (x,y) is of high proximity to obstacles} \\ 0 & \text{otherwise} \end{cases}$ | cf. Section 3.5 |
| | $c_{dynamic}^{obstacle}(x,y,t) = \begin{cases} \infty & \text{if (x,y, t) is in collision space} \\ \text{HIGH-COST} & \text{if (x,y, t) is of high proximity to obstacles} \\ cost_{distance}^{keeping} & \text{if (x,y, t) is in } follow \text{ area} \\ 0 & \text{otherwise} \end{cases}$ | cf. Section 3.5 |
| Lane keeping | $c_{lane-centering}(x,y) = \begin{cases} \infty & \text{if } (x,y) \text{ is in collision space} \\ C_{opp} + C_{dev}\lvert\delta l\rvert & \text{if } (x,y) \text{ is on lanes with oncoming traffic} \\ C_{dev}\lvert\delta l\rvert & \text{otherwise} \end{cases}$ | Punish deviation from the center of the current lane. cf. Section 3.5 |

Table 5.4: Cost functions applied in the proposed planner (part 1).

| General purpose | Cost function | Description |
|---|---|---|
| Trajectory feasibility | $c_{\alpha_{lon}}(\alpha_{lon}) = \begin{cases} C_{\alpha_{lon}}\lvert\alpha_{lon} - \alpha_{lon}^{soft-max}\rvert & \text{if } \alpha_{lon} > \alpha_{lon}^{soft-max} \\ C_{\alpha_{lon}}\lvert\alpha_{lon} - \alpha_{lon}^{soft-min}\rvert & \text{if } \alpha_{lon} < \alpha_{lon}^{soft-min} \\ 0 & \text{otherwise} \end{cases}$ | A penalty is given if the longitudinal acceleration of the trajectory exceeds the limits defined by the maximum and minimum accelerations allowed without penalty (cf. Section 4.4). |
| | $c_{\alpha_{lat}}(\alpha_{lat}) = \begin{cases} C_{\alpha_{lat}}\lvert\alpha_{lat}\rvert & \text{if } \alpha_{lat} \text{ is within } \alpha_{lat}^{limit} \\ \infty & \text{otherwise} \end{cases}$ | Within the limit of the lateral acceleration capable of the vehicle dynamics, the cost grows linearly with the amount of the lateral acceleration. Once the limit is exceeded, $\infty$ is assigned. |
| | $c_{\dot{\kappa}} = \begin{cases} C_{\dot{\kappa}}\lvert\dot{\kappa}\rvert & \text{if } \dot{\kappa} \text{ is within } \dot{\kappa}_{limit} \\ \infty & \text{otherwise} \end{cases}$ | Similar to the cost for the lateral acceleration. |
| | $c_{\kappa} = \begin{cases} C_{\kappa}\lvert\lvert\kappa\rvert - \lvert\kappa_{limit}\rvert\rvert & \text{if } \kappa \text{ exceeds } \kappa_{limit} \\ 0 & \text{otherwise} \end{cases}$ | The trajectory that exceeds the curvature limit of the vehicle gets punished. |
| Observing speed limit | $c_{speed-limit}(v) = \begin{cases} cost & \text{if } v > 0.99 v_{limit} \\ 0 & \text{otherwise} \end{cases}$ | The maximum speed is set a hair below the speed limit of the travelling lane. |

Table 5.5: Cost functions applied in the proposed planner (part 2).

It is necessary to take some specific measures here in order to guarantee a certain level of planning consistency. These issues will be further discussed in the next two subsections.

Although the scenario reasoning module has not yet been embedded in the current implementation of the planner, it is shown here that it can be seamlessly integrated into the planning strategy. As can be seen from the construction process of the spatial horizon (cf. Section 3.1), a sequence of edges encoding the road information can be obtained as a by-product of the spatial horizon construction. After the distance horizon confining the spatial horizon is reached, the search for the edges can go on without further construction and update of the *LATTICE-SEC-RAW*s. Given the edges that extend beyond the planning horizon, some long-term (from the perspective of the limited planning horizon) information can be gathered. Such information is used by the scenario reasoning module to set target speeds and schedule the weights of the cost functions. Following those instructions, the motion planner can make more far-sighted decisions.

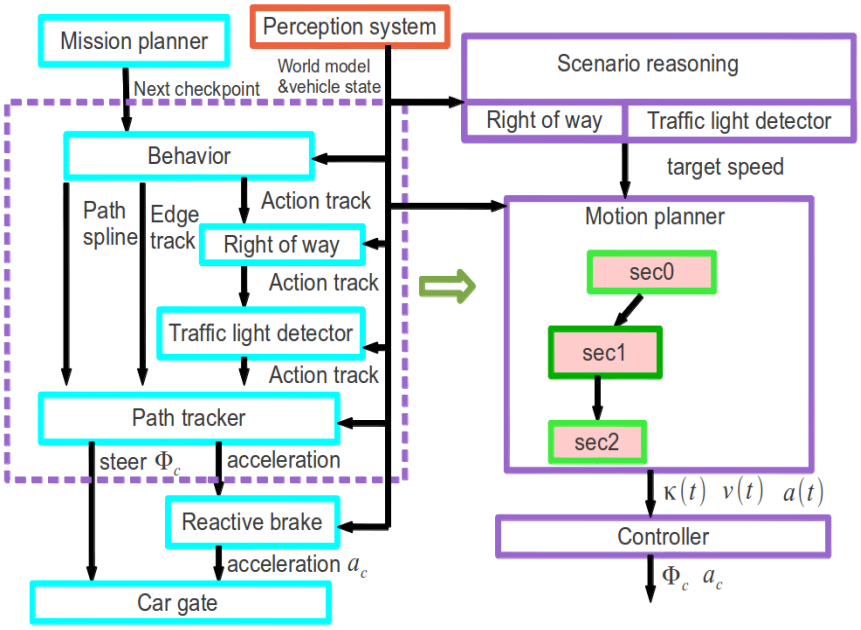



Figure 5.10: Planning system of MIG. Left: the existing planning architecture. Right: the proposed planner-based planning modules necessary for replacing the ones inside the dashed box on the left.

Figure 5.11 shows an example scenario related to this issue, where the information of the traffic light can be used to calculate a target speed for the planning within the planning horizon. Under the guidance of the target speed, the planner can generate a more smooth trajectory of deceleration.

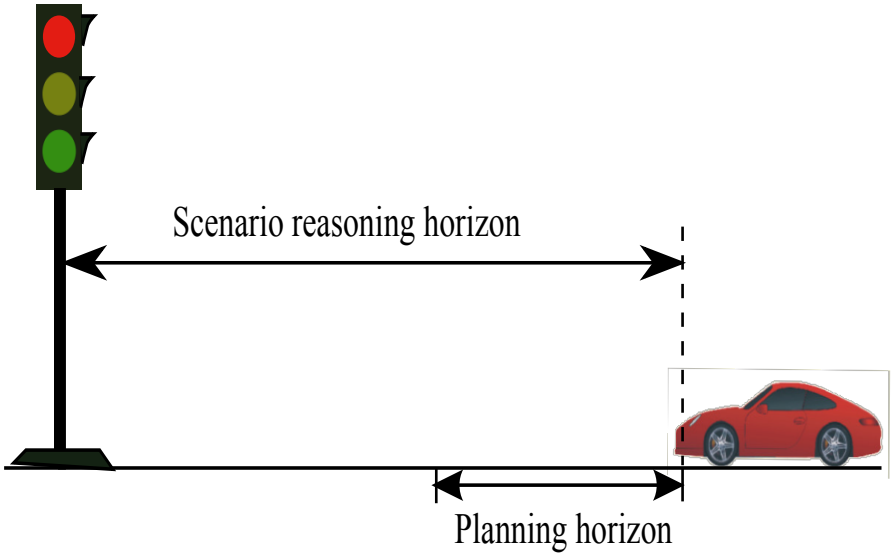



Figure 5.11: A driving scenario where the traffic light information extracted from the edges that extend beyond the planning horizon can assist the short-sighted planner in making far-sighted plans.

Given the sampled spatial horizon, the scenario-specific target speed and weights of the cost functions, the world model and the start state of the vehicle, it is ready for the planner to compute the best constraint-abiding trajectory. The resultant trajectory is sampled into a sequence of vector samples $\{(\kappa(t), v(t), \alpha(t))\}$. These vector samples are then sent to the controller where the actuation commands are calculated. Both the planner and the controller maintain a queue of trajectories sampled in this way. The planner uses the trajectory queue to perform latency compensation, as is described later in Section 5.3.2. The controller extracts the desired states $(\hat{\kappa}(t_{cur}), \hat{v}(t_{cur}), \hat{\alpha}(t_{cur}))$ from the most recent trajectory that is valid over the current time $t_{cur}$. Based on the desired states and the current states $(\kappa, v, \alpha)$ of the vehicle, the controller calculates the steering and throttle/brake actuation commands $(\phi_c, \alpha_c)$ according to:

$$\begin{aligned} \phi_c &= -arcsin(L\hat{\kappa}) \\ \alpha_c &= k_p(\hat{v} - v) + k_d(\alpha_{diff}) + k_{ff}\hat{\alpha} \end{aligned} \tag{5.3}$$

where $L$ refers to the distance between the vehicle's front and rear axles, $k_i, k_p$ are the parameters of the PI controller, $k_{ff}$ is the gain of the feedforward controller, and $\alpha_{diff}$ is the discrepancy between the desired acceleration $\hat{\alpha}$ and the predicted acceleration outcome. The prediction is based on the acceleration model identified in [88] and applied currently in the simulator. It can be seen from Equation 5.3 that the acceleration controller is a PI plus feedforward controller. The calculation of the steering actuation command $\phi_c$ follows a nonholonomic vehicle model, i.e, the bicycle model shown in Figure 5.12, which is also applied in the simulator for the simulation of the lateral movement of the vehicle. It is noteworthy that the trajectory generated by the proposed planner describes the expected locus of the middle of the front axle of the vehicle. That is the reason why $arcsin$, rather than $arctan$, is used in the calculation of $\phi_c$.

### 5.3.2 Latency Compensation

Let $\tau_{cur}$ denote the trajectory generated in the current planning cycle based on the current state $P_{start}$ of the vehicle. Let $t_{end}$ refer to the end of the planning cycle. By the time the planning cycle is finished and $\tau_{cur}$ is ready for the controller to execute, the vehicle has moved on to a new state $P_{end}$ by following the existing trajectories $\{\tau_{pre}\}$ produced from previous planning cycles. As a result, such situation as demonstrated in Figure 5.13(a) might happen, where $P_{end}$ has a large discrepancy from the state $P_{exp}$ on $\tau_{cur}$ at the time of $t_{end}$. Such inconsistency between trajectories generated from consecutive planning cycles poses a threat to tracking accuracy. Although the application of the cost functions for keeping trajectory consistency can help to mitigate this problem to some extent, it is still not sufficient. To deal with this issue, a forward simulation of the vehicle dynamics is implemented to estimate $P_{end}$ at the beginning of the planning cycle so that $\tau_{cur}$ can start from the estimated $P_{end}$ rather than $P_{start}$.

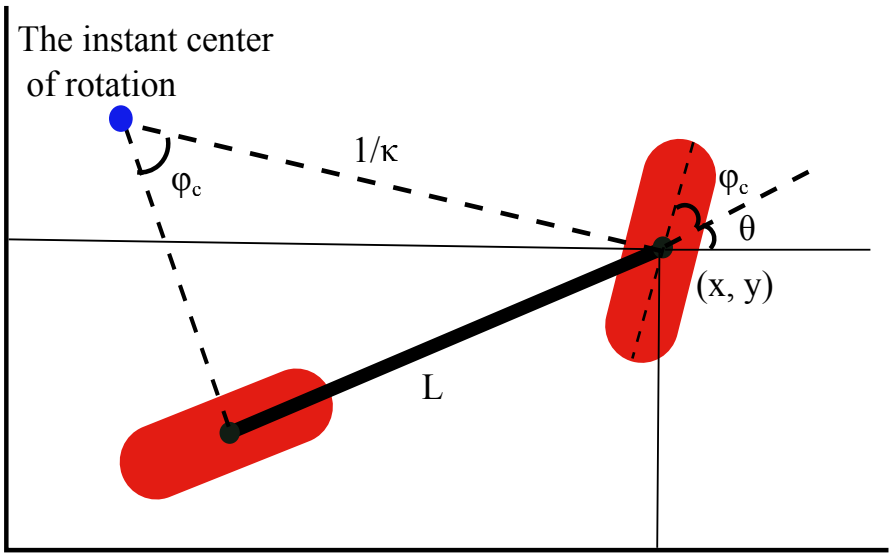


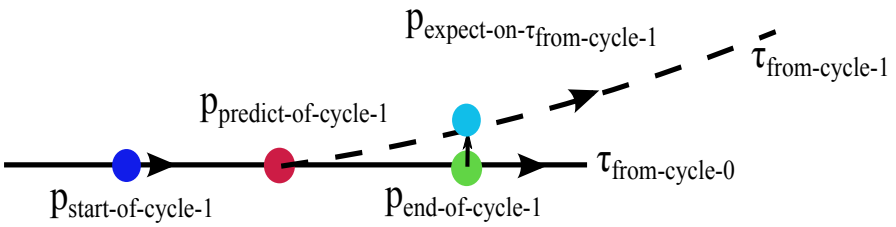

Figure 5.12: Bicycle model applied in the simulator for the calculation of the lateral movement of the vehicle.

The question following this measure is how far into the future the forward simulation should be conducted. As can be inferred from Figure 5.13, the principle in determining the simulation duration is that the simulation duration should not be shorter than the duration of the planning cycle. Nonetheless, due to the potential discrepancy between the resultant state of the forward simulation and the actual tracking result of the vehicle, it is still better to set the simulation duration as short as possible. In the current implementation, the simulation duration is predefined at the initialization stage of the planner and cannot be modified on the fly during the experiment. Potential heuristics need to be discovered to predict the duration of the planning cycle in runtime.

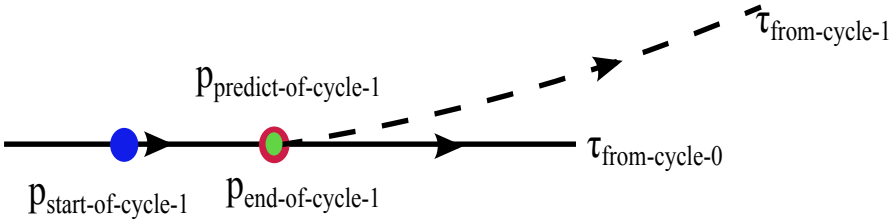
Based on the assumption that the vehicle control system executes the control commands correctly, the forward simulation uses the fourth order Runge-Kutta method (cf. [89]) to integrate the effects of the control commands on the vehicle state system that is illustrated in Equation 3.4. The forward simulation extracts the control commands from the reserved trajectory queue in the same way as the controller. To keep consistency between the forward simulation and the execution of the controller, a timestamp $t_{exe}$ is attached to each newly generated trajectory $\tau_{cur}$, which denotes the time when $\tau_{cur}$ is executed by the controller for the first time. The timestamp $t_{exe}$ is given as:

$$t_{exe} = max\{t_{start}(\tau_{cur}), t_{end}(cycle)\} = max\{t_{start}(cycle) + \delta t_{latency}, t_{end}(cycle)\} \quad (5.4)$$

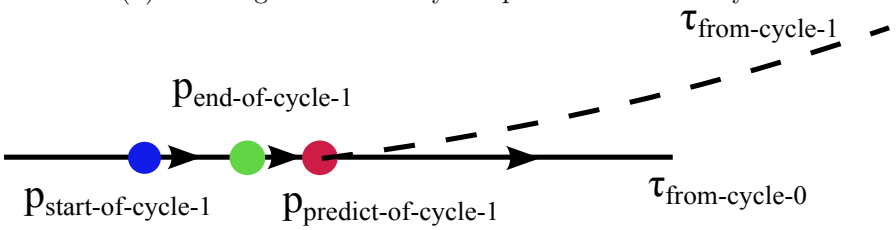
where $t_{start}(cycle)$ is the starting time of the planning cycle, $t_{start}(\tau_{cur})$ is the predicted ending time of the planning cycle with which the new trajectory $\tau_{cur}$ starts, $\delta t_{latency}$ is the assumed amount of latency, and $t_{end}(cycle)$ is the actual ending time




(a) Planning with insufficient compensation for latency.




(b) Planning with necessary compensation for latency.




(c) Planning with more than necessary compensation for latency.

Figure 5.13: Potential effects of different amounts of compensation for latency on trajectory consistency.

of the planning cycle. The time $t_{exe}$, rather than the more straightforward $t_{start}(\tau)$, is taken by the forward simulation as the time when the trajectory comes into effect. It is assumed in the current implementation of the proposed planner that the trajectory will be executed by the controller once it is generated and becomes the most recent trajectory that is valid over the time of the execution. In this way, the behaviour of the controller in terms of the extraction of control commands from the reserved trajectory queue keeps in accord with that of the forward simulation.

It should be pointed out that the proposed strategy of latency compensation does not take into account the actuation latency, i.e., the delay between the output of the actuation commands from the controller and the reaction of the vehicle to those commands. It is mentioned in [23] that such latency is not ignorable. This issue should be addressed in future work.

### 5.3.3 Spatial Lattice Consistency

The spatial horizon is specified according to the predicted vehicle state resulting from the forward simulation. In order to promote trajectory consistency between consecutive planning cycles, it should be first guaranteed that the remaining segment of the trajectory generated from the last planning cycle can still be constructed in the current planning cycle as suggested by the definition of temporal consistency (cf. Definition 3.1).

Figure 5.14 demonstrates the importance of keeping spatial consistency. To that end, the spatial lattices of consecutive spatial horizons should keep consistent as much as possible. That is, suppose there is a lattice that is constructed once for the complete travel and thus fixed to the static road network, the spatial lattice for an arbitrary planning horizon can be regarded as one part of the complete lattice. In other words, the spatial lattice does not slide along the road network with the vehicle. Recall that the state lattice is defined by four elements, i.e., latitude sampling interval($a_l$), station sampling interval ($a_s$), spatial horizon and lateral reference (i.e., the set $\{(s, 0)\}$). As $a_l$ and $a_s$ are constant during the whole travel, they do not affect lattice consistency. Consequently, there are two elements left, i.e., spatial horizon and lateral reference. Recall that a Spatial lattice is a discrete representations of the spatial horizon. By definition, the first station sample $s^0_{lattice}$ of the corresponding spatial lattice is the starting station $s^0_{horizon}$ of the spatial horizon. Since consecutive spatial horizons are consistent by definition ( cf. Section 3.1), only its starting station $s^0_{horizon}$ has a role in deciding lattice consistency. In the proposed planner, the strategy adopted for specifying $s^0_{horizon}$ is as follows:

- Map the predicted vehicle position $(x, y)$ to the state lattice generated in the last planning cycle, if it exists. This mapping can be readily achieved by using the $XYSL$ map generated in the last planning cycle, i.e., $(s_{start}, l_{start}) = XYSL(x, y)$.

- As the coordinate $s_{start}$ may not coincide with one station sample of the last lattice, it should be further rectified. Let $\hat{s}$ denote the rectified coordinate. It is calculated as:

$$\begin{aligned} s_i &= \lfloor \tfrac{s_{start}}{a_S} \rfloor \\ \hat{s} &= s_i a_S \end{aligned} \tag{5.5}$$

  where the notation $\lfloor\ \rfloor$ is used to get the greatest integer that is not larger than the quantity inside it. Finally, $\hat{s}$ is assigned to $s^0_{horizon}$.

In this way, the spatial lattices of consecutive spatial horizons are consistent in terms of the discretized station. The remaining element of the spatial lattice, i.e., the lateral reference, determines the consistency of lattices in terms of the discretized latitude. Recall that one section of the lattice (i.e., *LATTICE-SEC*) may contain several parallel lane segments. In such case, the proposed planner defines the center of the lane which covers the starting position of the vehicle to be the lateral reference of the lattice. That is, this lane center is sampled and the $l$ coordinates of its samples are zero. Recall that a *LATTICE-SEC* also records the center lines of all the lanes that appear within it. Consequently, the center line $cl_0$ of the current lattice is determined according to the following steps:

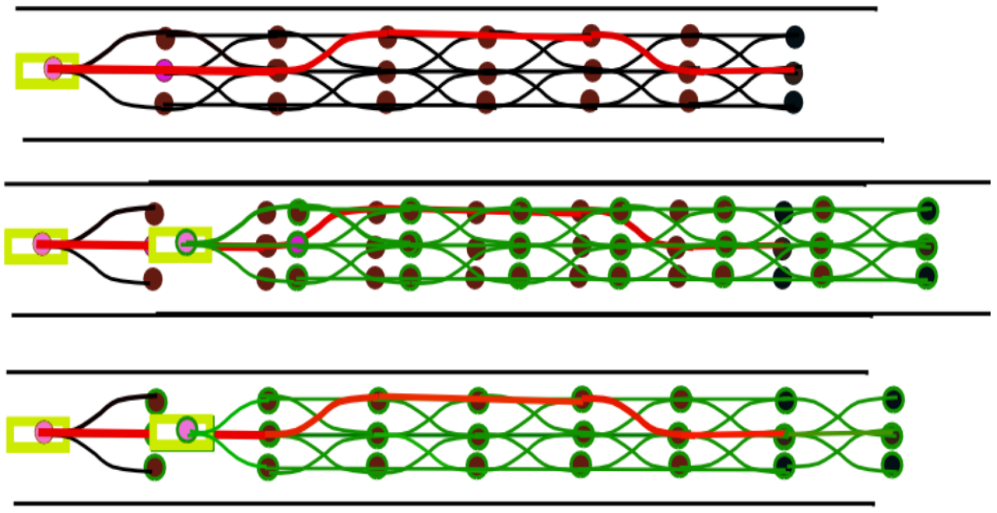


Figure 5.14: A demonstration of the importance of keeping spatial consistency. The first figure shows the first spatial planning horizon (black) and a trajectory selected (red). At the beginning of the second planning cycle, the car moves to a new position. If the subsequent planning horizon starts simply from the position of the car (as is shown in the second figure, green), then the plan generated from the last planning horizon cannot be built in the new planning horizon. However, if the subsequent planning horizon starts from one layer of the last planning horizon (as is shown in the third figure,green), then the remaining segment of the last plan can still be found.

- Get the *LATTICE-SEC* of the last lattice whose domain of station covers $s^0_{horizon}$. Let this *LATTICE-SEC* be $sec_0$.
- Find out the lane segment $lane_0$ in $sec_0$ whose domain of latitude covers $l_{start}$. The center of $lane_0$ is chosen to be $cl_0$ to facilitate the lane centering behaviour of the vehicle.

The starting point $p_0$ on $lane_0$ for the search of edges is defined by $cl_0$ and $s^0_{horizon}$. Given $p_0$, the spatial horizon can be readily constructed and sampled by following the algorithms presented in Section 3.1. The spatial lattices defined in this way are consistent in most cases. The case where lattice inconsistency might appear is when a lane change manoeuvre takes place. As is demonstrated in Figure 5.15, the lateral reference of the second lattice is not the same as the one adopted in the first lattice. As the proposed planner samples the spatial horizon uniformly along the lateral direction, it might happen that the center lines of the non-reference lanes are not sampled. The situation of the lane on the top of Figure 5.15 from the perspective of the first lattice is just like that. Once the center of such a lane is chosen to be $cl_0$ of the new lattice, which happens when the vehicle drives onto this lane, a lattice inconsistency occurs. As can be seen from Figure 5.15, the consequence of such lattice inconsistency is a trajectory inconsistency

which will cause tracking errors. One approach to deal with it can be adopting non-uniform sampling along the lateral direction to make sure that all the center lines within the spatial horizon are sampled.

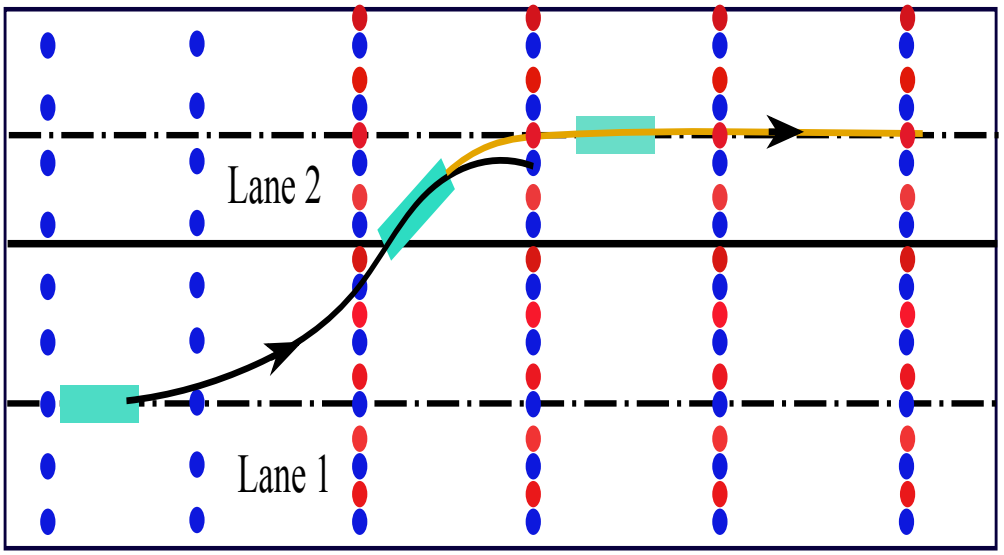



Figure 5.15: Lattice inconsistency caused by the lane change manoeuvre. The blue points are the spatial nodes sampled laterally in reference to the center of the first lane, and the red points the center of the second lane. The black curve is the plan generated based on the blue points, which is inconsistent with the orange curve that represents the trajectory constructed based on the red points.

## 5.4 Summary

This chapter described how the proposed motion planning algorithms presented in previous chapters are employed to embody a concrete motion planner. The inexpensive modules of the planner are implemented on the CPU, while the costly procedures, i.e., the evaluation of the paths and the construction of the state lattice, are conducted on the CUDA-enabled GPU. The details of the implementation were presented, including the update of the spatial lattice, the storage of the path edges, the construction of the $XYSL$ map to facilitate the generation of cost maps, the storage of the cost maps, the execution of the trajectory generation kernel, the interactions between the threads and the memory segments where the relevant data are stored, the selection of the best target node and the reconstruction and sampling of the best constraint-abiding trajectory. After the presentation of the implementation of the core planner, the cost functions implemented in the proposed planner were listed.

The integration of the concrete planner into the existing planning system of MIG was also displayed. This task is non-trivial as the existing planning architecture is built based on a path planner. Consequently, some of the existing modules need to be rearranged and some new models are required to be constructed, such as the controller that is

necessary for executing the motion plans. Besides, it was demonstrated that a latency compensation is necessary for achieving trajectory consistency. Lattice consistency was also promoted for the same purpose.

So far, the algorithms and implementations of the proposed planner were presented. As the planner is evaluated in a simulation environment, the simulation of the scanning sensors is necessary for a more realistic experiment environment. In the next chapter, the details about the sensor simulation are presented. The evaluation of the proposed planner is reported in Chapter 7 after the sensor simulation is illustrated.

# Chapter 6

# Scanning Sensor Simulation

A simulation with a high realistic level can make the test of autonomous vehicles time-efficient, cost-effective and safe. As perception insufficiency and inaccuracy challenge the flexibility and robustness of the motion planner, the simulation of scanning sensors is indispensable for an effective simulation environment for testing the motion planners. This chapter presents a GPU-based approach to simulate scanning sensors. Above all, Section 6.1 gives a general introduction of the behaviour and performance of real-life scanning sensors, based on which the sensors are modelled in the simulation. Following an introduction of the software employed for the sensor simulation in this work, Section 6.2 illustrates the GPU-based implementation of the sensor simulation. Besides, potential errors introduced by the application of graphics rendering pipeline are explained and a macro-micro method is proposed to mitigate the problem. Finally, Section 6.3 reports the performance of the scanning sensor simulation strategy in terms of accuracy and time efficiency. The old version of the scanning sensor simulation can be found in [90]. The simulation method illustrated in this chapter is an adapted version based on the old one.

## 6.1 Scanning Sensor Modelling

This section begins with a general introduction to the physical properties of the real-life scanning sensors. Based on that introduction, the problems concerning the simulation of the scanning sensors are discussed. After the specific properties and outputs of the scanning sensors employed by the autonomous vehicle system are illustrated, the simulation models of the sensors, together with the assumptions about those models, are provided. The working process and configuration parameters of the simulated sensors are presented.

### 6.1.1 Introduction to Real-life Radar and LiDAR

Radar (Radio Detection And Ranging) is a device that emits radio waves to the surroundings and receives the reflected echoes. Radio waves are characterized mainly by

their frequency, speed and intensity. These properties together determine the energy of the radio signal. Several physical phenomena can happen during the propagation of the radio waves, such as reflection, refraction and attenuation. Reflection includes backscattering of homogeneous surfaces and specular returns from plane surfaces, e.g., water and man-made objects. Refraction takes place when the radio waves pass through the interface of two media with different refraction indices. The intensity of the radio wave may be attenuated by the media and other objects encountered during its transmission. The frequency of radio waves will change when relative radial movements (approaching and receding) exist between the radar sensor and the reflective object. Such phenomenon is referred to as Doppler effect. The location of the reflective object can be determined either through an analysis of the energy of the returned signals or based on their time of flight. For the kind of radars that relies on the energy analysis, the velocity of the radar sensor relative to the reflective object can be obtained based on the analysis of Doppler effect. For more details about the physical properties of real-life radars, please refer to [91].

LiDAR (light detection and ranging) works in a similar way to radar from the perspective of the way they send and receive signals. The signals transmitted by LiDAR are lasers (Light Amplification by Stimulated Emission of Radiation). In general, the distance of the LiDAR sensor relative to the reflective target is determined by the time of flight of the lasers. There are also cases where energy analysis is adopted to calculate the location of the target. Lasers demonstrate similar physical phenomena to radio waves as both kinds of signals belong to the family of electromagnetic signals. There is a detailed description of the working principles of LiDAR sensors in [92].

### 6.1.2 Radar Modelling

It can be concluded from the introduction presented above that the most important thing in simulating scanning sensors is modelling the propagation of the radio waves or lasers, including their interactions with the surrounding environment. Given all the necessary physical equations concerning the behaviours of the signals, sensor simulation that takes into account the complete set of physical phenomena is still challenging. For one thing, it is not easy to provide a detailed description of the surrounding environment in terms of all the physical properties related to the performance of the sensors. For another thing, tracing the complete trajectory of the signal and calculating all its interactions with the surrounding environment require an intolerable amount of computational resources, which makes it hard to satisfy the high demand from real-time applications on computational efficiency. Nonetheless, efforts can still be made to explore the potential of full-featured simulation. This is left for future work.

The simulation of the scanning sensors presented in this work is intended to provide

real-time sensor data for obstacle detection and tracking modules. It is more input-output oriented than physics oriented. Given a specific traffic scenario as the input, the simulated sensor is designed to give the same output as an ideal sensor will do. By *ideal*, it is addressed that the sensor can provide exact information of the reflective targets from its point of view. In this sense, no noise model is embedded in the simulation, which requires further improvements in future work. The radar system mounted on MIG has a post-processing module itself. That is, the radar system is responsible for analysing raw radar signals, and the final output from the radar system contains only the velocities and distances of the targets relative to the radar sensor. As a result, it is not necessary to simulate the radar signals. Accordingly, the radar model applied in this thesis works as follows:

- Locate the object targets that fall within the field of view (FOV) of the radar sensor.
- Calculate the relative distance and velocity between each of the targets and the radar sensor based on their absolute velocities and distances.
- Generate other relevant data about the scenario that would be recorded by a real-life radar sensor given the traffic scenario.
- Wrap the information obtained in the previous two steps in a package. The format of the package is in accord with the actual message sent out from the real-life radar sensor.
- Send the package to the obstacle detection and tracking modules for further processing. Repeat the whole process.

Important configuration parameters of the radar model include:

- Maximum detection ranges for targets like passengers, cars and trucks.
- Maximum horizontal field of view (HFOV).

### 6.1.3 LiDAR Modelling

The problems of simulating the scanning sensors discussed above are also encountered in the simulation of LiDAR sensors. At the moment, the model of the LiDAR sensor also mimics the ideal LiDAR. That is, the simulated LiDAR sensor can only provide static and ideal sensing results without considering the related dynamic physical process and potential noises.

There are two kinds of LiDAR sensors equipped on MIG. As can be seen from Figure 1.1, one type is the HDL-64E Velodyne sensor mounted on the top of the vehicle with an HFOV of 360°; the other type is the IBEO LUX LiDAR sensor, from which

six are installed around the vehicle. The layout of the transmitters and receivers of the Velodyne sensor is demonstrated in Figure 6.2. Figure 6.1 shows the four-layer laser model of the IBEO sensor. The lasers of the Velodyne sensor rotate around its vertical axis, while those of the IBEO sensor oscillate about its vertical axis. The HFOVs of the sensors are rendered by the rotation or oscillation. It might be straightforward to simulate the rotation or oscillation of the LiDAR sensors stepwise following the rate of change of the orientations of the lasers and gather the sensor data at each step. However, that is hard to achieve as the rate of change of the orientations of the lasers is very high. For example, in the case of the Velodyne sensor with a rotation rate of 5 HZ and an angular resolution of $0.09°$, the rate of change of the orientations of the lasers amounts to $20,000/s$, which is impossible to achieve in the simulation on general-purpose computers. As a result, such dynamic rotation behaviour is not considered in the simulation; instead, the HFOV rendered by the rotating lasers are assumed to exist all the time. In other words, it is assumed that the LiDAR sensor fires the laser beams at even horizontal angular intervals within its HFOV simultaneously. The value of the angular interval is specified based on a trade-off between the requirements of computational efficiency and simulation accuracy.

The working process of the LiDAR simulation is similar to that of the radar simulation. The only difference lies in that the output of the former consists of the distances and basic classifications (e.g., ground, dirt and rain) of the targets and the intensities of the returned laser signals. The distance and classification can be easily generated given the information of the sensor and the surrounding environment. The intensity of the echoed signal is affected by many factors such as the incidence angle of the laser with respect to the target surface, the color and material of target, rain and snow in the propagation medium, etc. The calculation of the intensity can be regarded as one part of a full-featured physical simulation which is difficult to implement for the reasons illustrated previously. Furthermore, the intensity property of the returned laser is still not used in the currently implemented obstacle detection and tracking modules. Consequently, the intensity is not calculated in the current simulation.

Important configuration parameters of the LiDAR sensor models are as follows:

- The maximum detection range.
- The HFOV and VFOV.
- The vertical and horizontal angular resolutions.

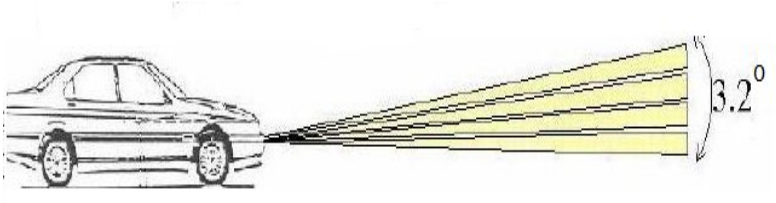



Figure 6.1: Four scanning layers of the IBEO Lux LiDAR (3.2°$VFOV$ by 110°$HFOV$), from [93]

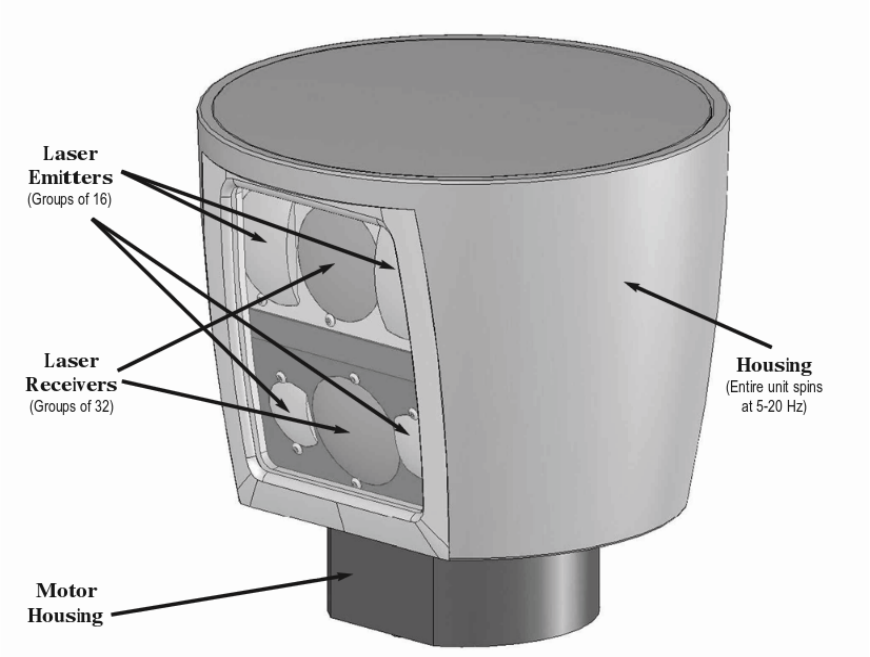



Figure 6.2: Design overview of HDL-64E Velodyne, from [94]

## 6.2 Implementation of the Scanning Sensor Simulation on the GPU

In general, the lasers and waves of scanning sensors work in a similar way to the visible light waves from the perspective of image rendering. The main difference lies in that the latter carries color information rather than information like distance and velocity as in the case of the former. Accordingly, image rendering hardware can be exploited to facilitate easy implementation of the simulation and relatively fast generation of sensor data. The GPU (graphics processing unit) is a device designed particularly for accelerating graphics rendering and is available on almost all modern computers. Moreover, most GPUs support software mechanisms that allow a certain level of customization of the rendering pipeline. This work uses one of such mechanisms, i.e., OpenGL shader, to realize the simulation of scanning sensors. This section presents the implementation to the simulation on the GPU. Before the presentation, a general introduction of the related software is provided.

### 6.2.1 OpenGL and OpenSceneGraph

OpenGL, developed by SGI (Silicon Graphics Inc.), is an API (Application Programming Interface) providing low level instructions and data standards for graphics rendering and is supported by almost all modern graphics hardware [95]. Figure 6.3 shows the geometry rendering pipeline based on the OpenGL rendering standards. By default, the *per-vertex operation* transforms the geometric data, such as vertex positions and normal vectors, from object space to eye space. In the coordinate system of eye space, the virtual camera is located at the origin and facing to $-z$ axis as is demonstrated in Figure 6.4. If lighting is enabled, the lighting calculation per vertex is performed, and the colors of the vertices get updated. At the stage of primitive assembly, the geometry primitives are assembled and further tessellated into simple primitives (point, line, triangle). The clipping operation transforms the primitives from eye space to clipping space and clips the primitives which cross the clipping planes defined by the viewing frustum. The transformation can be either a perspective projection or an orthographic (parallel) projection. The basic idea of the perspective projection is shown on the right of Figure 6.4. Primitives that are hidden by other objects from the perspective of the camera or fall outside of the viewing frustum are discarded. The primitives that survive the stage of clipping and culling are then rasterized. The result of the rasterization is a sequence of *fragments*. A fragment is a set of states including its position in screen space and arbitrary data pertaining to the vertices that were output from previous stages. The data of the fragment are calculated by interpolating between the data values of the vertices for the fragment in question. The fragment data are further processed in the fragment processing stage. The data such as color, depth and stencil values will be written into frame buffers should they survive several tests such as the stencil and depth tests.

As is demonstrated in Figure 6.3, some processing modules (framed in dashed boxes) along the pipeline are customizable via shaders. Shaders are compilation units designed to define the operations of the processors at the programmable stages of the rendering pipeline. The scanning sensor simulation proposed in this thesis uses vertex and fragment shaders written in the OpenGL Shading Language (GLSL) (cf. [96] for more details).

OpenSceneGraph (OSG) is a high level 3D graphics programming toolkit based on OpenGL [97] [98]. It hides the low level execution details of OpenGL from the programmer and thus allows a relatively easy scene rendering programming. OSG provides a hierarchical data structure called *scene graph* allowing programmers to describe what to draw; its rendering engine then traverses that scene graph and signals the graphics hardware to execute the underlying rendering commands [95]. Figure 6.5 gives an example of such scene graph. OSG has a *camera* class which provides a convenient interface for the program to manipulate the rendering procedures. An object of the camera class is regarded as a node from the perspective of the scene graph. Through camera nodes, the

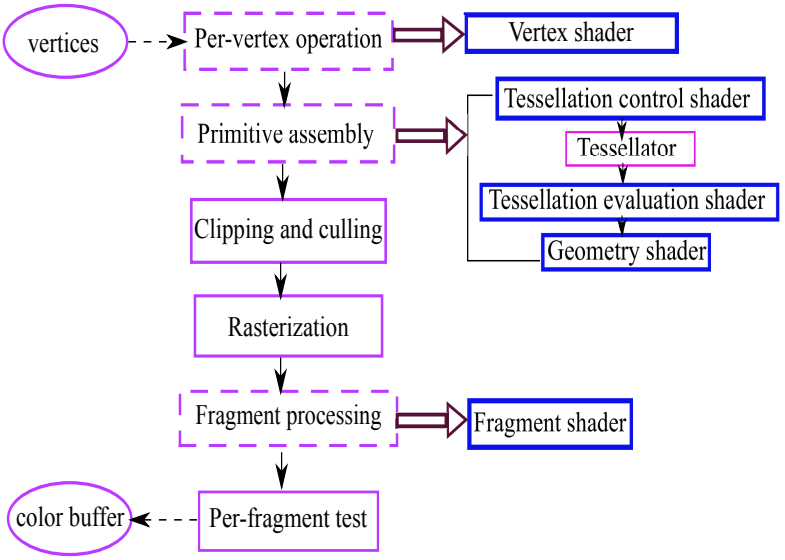



Figure 6.3: OpenGL geometry rendering pipeline ( cf. [96] for more details).

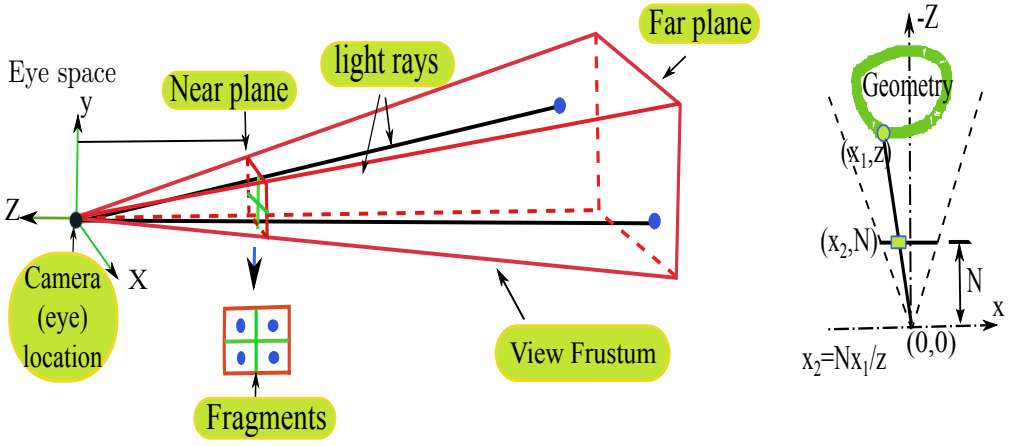



Figure 6.4: Eye space and perspective projection.

program can output geometric data to the graphics hardware and read out the rendered images from frame buffers on the hardware. The program can define the programmable stages along the rendering pipeline by loading shaders via camera nodes. All the parameters used to define the viewing frustum (cf. Figure 6.4 ) and other parameters related to the rendering pipeline can be set via the camera node.

### 6.2.2 Shader-based Scanning Sensor Simulation

The proposed simulation strategy uses camera nodes provided by OSG to configure the rendering pipeline implemented on the GPU. Figure 6.6 demonstrates the work flow of the sensor simulation. The simulator is responsible for the simulation of the dynamics of the autonomous vehicle. As the scanning sensors are fixed on the vehicle body, their positions and orientations change all the time according to the position and orientation of the vehicle. The positions and orientations of the camera nodes keep in accord with the

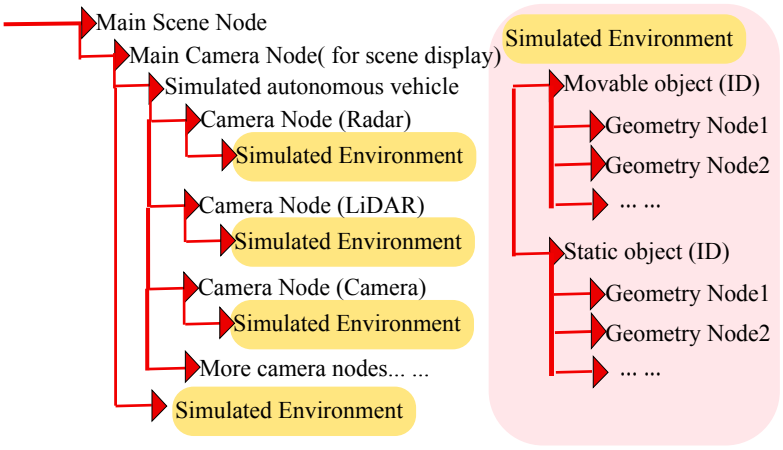



Figure 6.5: Scene graph with objects in the traffic environment and sensor models.

sensors and thus get updated each time when the vehicle pose changes in the simulator. The *dummy-controller* is in charge of updating the positions and orientations of all the objects in the traffic scenarios. The camera node configures the rendering pipeline using the sensor information provided by the simulator and sends the updated data of the traffic participants to the GPU. In the rendering process, vertex and fragment shaders designed to capture the sensor-specific information replace the default modules that are used to record the color information. The behaviours of the simulation-oriented vertex and fragment shaders for the simulations of LiDAR and radar sensors are shown in Figure 6.7 and Figure 6.8 respectively. The position correction based on the micro method mentioned in Figure 6.7 will be discussed later. It is noteworthy that each object in the traffic environment has a unique identity (ID). The ID of the object is assigned to the object node in the scene graph. In this way, all the vertices of the object node get the same ID pertaining to this object. Such information as the object ID can be obtained by the shaders through the vertices. From the perspective of the CPU, each object ID corresponds to a set of information pertaining to the object in question, including:

- Maximum detection distance for radar.
- Basic classifications (e.g., ground) for LiDAR.
- Object velocity for radar.

Once the images encoding the sensor-specific information are rendered to the frame buffer, the program running on the CPU can read them out. The object ID recorded in each valid fragment can then be used to query the properties of the objects that are stored on the CPU. After that, the sensor data generation modules compile the simulated sensor data in packages consistent with the actual sensor messages. Finally, these packages are sent to the obstacle detection and tracking modules for further processing.

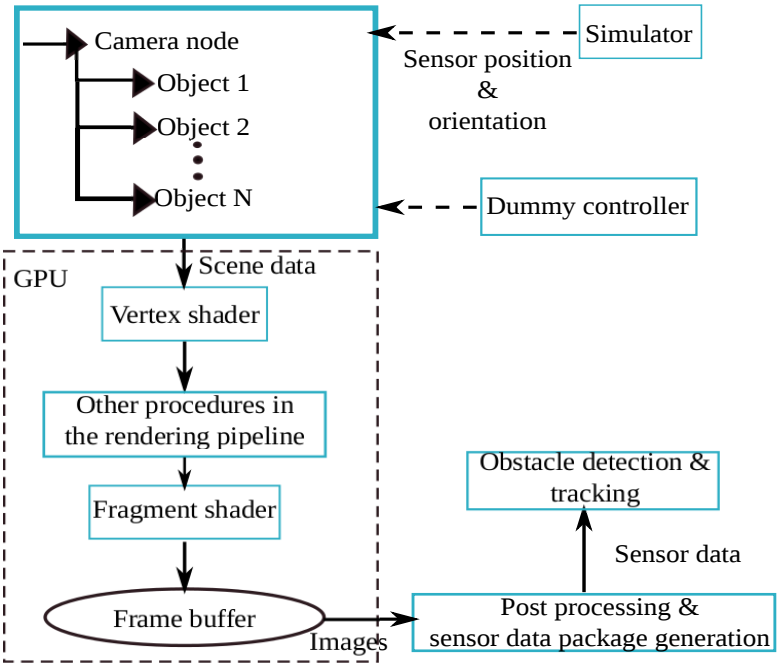



Figure 6.6: Work flow of the sensor simulation.

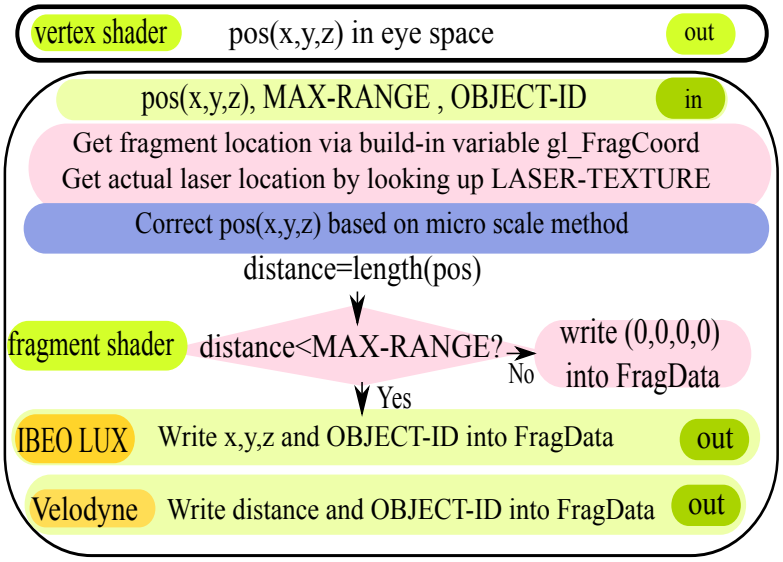



Figure 6.7: Vertex and fragment shaders for LiDAR simulation.

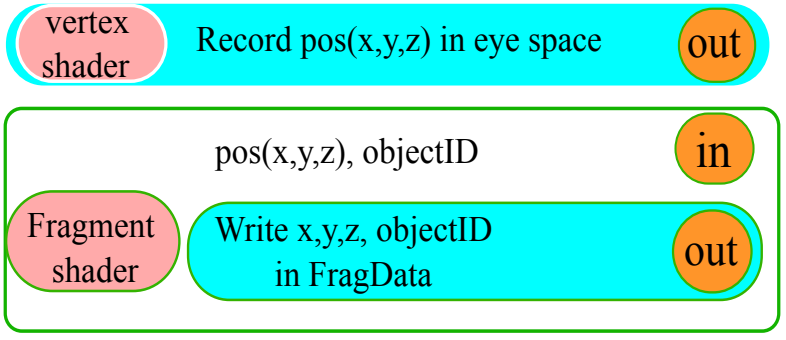



Figure 6.8: Vertex and fragment shaders for radar simulation.

### 6.2.3 Mitigating Errors Caused by Inconsistency between Lasers and Virtual Light Rays

The lasers of real-life LiDAR sensors rotate or oscillate around an axis, as is demonstrated in Figure 6.9. As a result, the lasers that are not perpendicular to the rotation axis trace out conical surfaces, as is demonstrated in Figure 6.10(a). In comparison, the light rays used to simulate the lasers in the viewing frustum, i.e., those virtual rays that pass through the centres of the pixels on the near plane, render a plane surface (cf. Figure 6.10(b)). As the HFOV becomes larger, the mismatch gets more obvious and cannot be ignored. Figure 6.11 gives an idea of such discrepancy. Recall that the lasers covering the overall HFOV are simulated at the same time. Consequently, the simulation of the lasers using one default camera node would cause intolerable errors. To that end, a macro-micro approach intended to deal with this problem is proposed as follows:

- Macro-scale approximation: approximate the laser beams with the light rays passing through the pixels that are nearest to the intersections of the laser beams and the near plane. The camera node used to simulate one LiDAR sensor has a much finer resolution than the sensor does itself. For example, the IBEO sensor expected to have $4 \times 220$ laser beams is simulated with one camera node which can render an image of $200 \times 400$ pixels. The FOVs of the IBEO sensor and the viewing frustum defined by the camera node are the same. Then the real-life laser beams are intersected with the near plane of the camera node, and the coordinates of the intersections are calculated. For each intersection, the pixel that is nearest to it on the near plane is located and recorded. The final result of the macro-scale approximation is a texture $LASER-TEXTURE$ containing $200 \times 400$ texels. Each texel is related to a unique pixel of the image that will be rendered by the camera node. If a pixel turns out to be nearest to the intersection of a laser beam and the near plane, its corresponding texel will record the coordinates of the intersection. Figure 6.12 displays this approximation scheme.

- Micro-scale correction: further rectify the target positions captured by the light rays chosen in the macro-scale approximation stage so that they can more closely approximate the information returned by the actual lasers. In the OpenGL rendering pipeline, the camera node is supposed to render the image onto the window screen. The window-relative coordinates of the image pixels (i.e. the fragments) can be obtained in the fragment shader via the built-in variable $gl\_FragCoord$ of the OpenGL shading language. The window-relative coordinates of the intersections of the actual lasers and the near plane can also be computed by scaling their coordinates on the near plane by the scale factor between the near

plane and the window screen. As mentioned above, the coordinates of the intersections on the near plane are already precomputed and stored in a texture $LASER-TEXTURE$. This texture is attached to the camera node as a uniform variable which can be accessed by the fragment shader. The partial derivatives of the data (in this simulation context, i.e., the rasterization result of the points on the reflective targets for the fragment) contained in a fragment with respect to the window-relative coordinates of the said fragment can be obtained by the built-in functions $dFdx()$ and $dFdy()$. Consequently, the data pertaining to the actual lasers can be approximately computed as:

$$\begin{aligned} pos_{actual} = \quad & pos_{pixel} + dFdx(pos)(x_{actual} - gl\_FragCoord.x) \\ & + dFdy(pos)(y_{actual} - gl\_FragCoord.y) \end{aligned}$$

where $pos_{actual}$ refers to the approximate position of the point on the reflector targeted by the actual laser beam that passes through the intersection, and $x_{actual}$ and $y_{actual}$ compose the window-relative coordinates of the intersection. The positions rectified in this way are much closer to the target positions captured by the real-life lasers than the default light rays are. Figure 6.13 shows the effect of this correction strategy.

There is still another problem regarding the simulation of the Velodyne sensor. Although the lasers of the Velodyne sensor are turning around the same axis, they are sent out from locations on the sensor with irregular horizontal and vertical offsets between them. For example, two laser transmitters can have a horizontal offset of 17 $cm$. Such laser layout cannot be efficiently modelled by camera nodes as the lasers simulated using the same camera node are supposed to have the same origin. In the current application, it is assumed that all the lasers sent out from the Velodyne sensor have the same origin. In this way, the lasers can be simulated with as few camera nodes as possible. The configuration file of the Velodyne sensor which stores the values of the original offsets is also modified according to that assumption. In this way, the Velodyne sensor data processing module can interpret the simulated sensor data correctly.

## 6.3 Performance Evaluation

This section evaluates the performance of the proposed scanning sensor simulation strategy. The GPU and CPU installed on the computer used for the evaluation are an Nvidia GeForce Quadro 2000 containing 192 parallel processing cores and an Intel Xeon E31245. The 3D models that compose the traffic environment consist of 16 houses, 1 road, 7 trees, 25 randomly on road running cars, 1 pedestrian and 6 traffic lights. The average size of

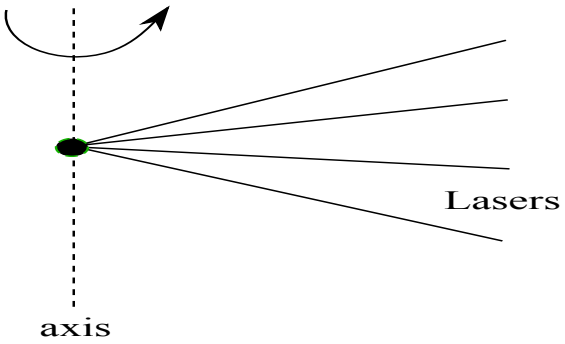
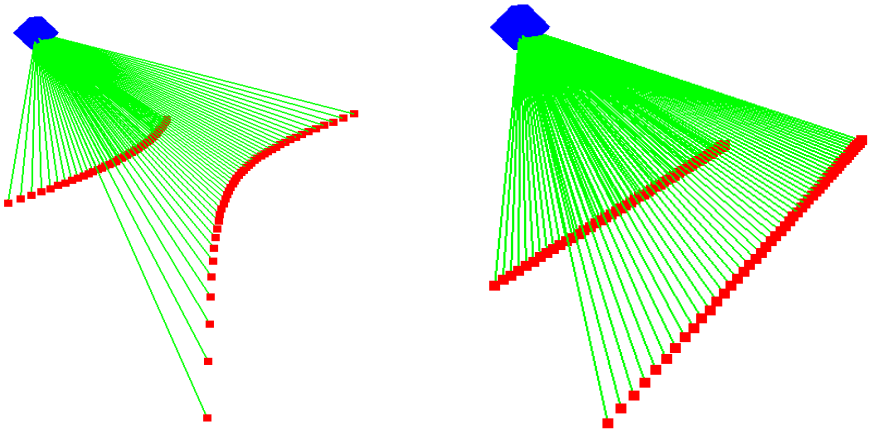



Figure 6.9: Four laser beams turning around an axis.



(a) Conical surface traced out by the actual lasers.

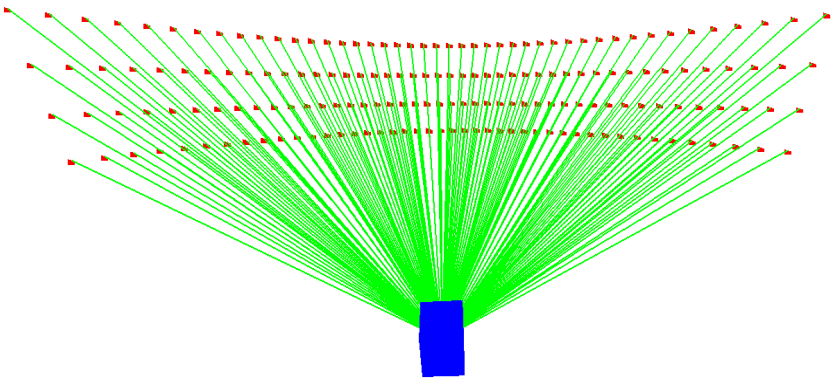
(b) Plane surface traced out by the light rays passing through the centres of the pixels of the rendered image.

Figure 6.10: Surfaces rendered by the actual lasers and the light rays within the viewing frustum.

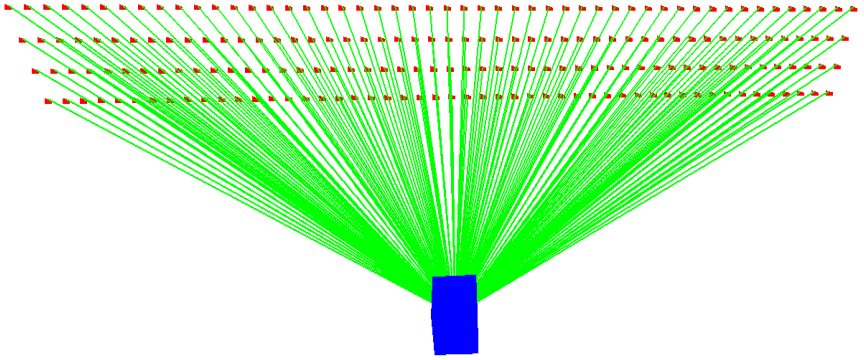
a 3D model file is $2MB$. The simulated sensors include 6 IBEO LiDAR sensors, each being simulated with 1 camera node, 1 Velodyne sensor simulated with 3 camera nodes and 1 radar sensor simulated with 1 camera node. For the corresponding videos, please refer to Appendix 1.

The accuracy of the simulated sensor data in comparison with the real sensor data has not yet been conducted, which is left for future work. At the moment, the accuracy of the simulated sensor data is only checked by visual display of the simulated sensor data and the objects identified by the obstacle detection module based on those data. The scanning results of the simulated IBEO, Velodyne and radar sensors are displayed in Figure 6.14, Figure 6.15 and Figure 6.16, respectively.

The computation time consumed by the rendering of the simulated IBEO sensors is also examined. The frame rates of the GPU for rendering simulated sensors with different configurations are reported in Table 6.1. As can be concluded from Table 6.1, the frame



(a) Scanning pattern of the actual lasers.



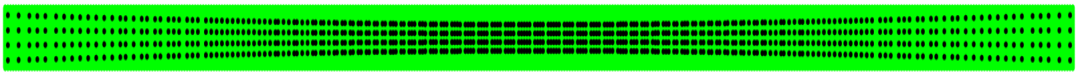
(b) Scanning pattern of the light rays passing through the centres of the pixels of the to-be-rendered image.

Figure 6.11: Scanning patterns of the actual lasers and the light rays within the viewing frustum. $VFOV = 20°$.

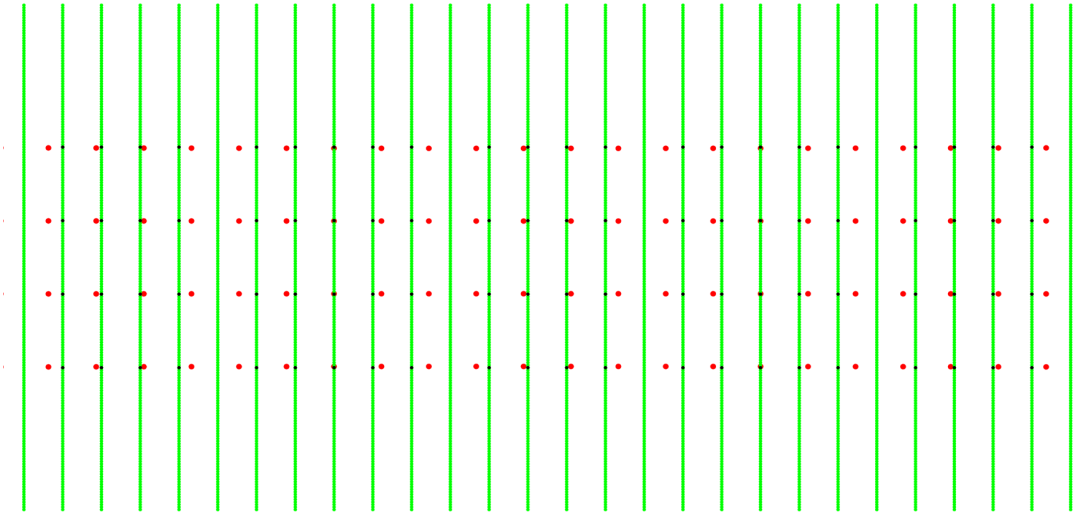
rate declines when the number of objects in the scene increases. Furthermore, the frame rate also drops drastically when the number of pixels used to simulate the sensor gets larger. Note that more accurate sensor data should be generated by a simulation with more pixels. In the case of the simulation of the IBEO LiDAR sensor, the size of pixels is set to be $200 \times 400$, which can generate sufficiently accurate sensor data (from the perspective of approximating the lasers with the virtual light rays). The corresponding computational cost is tolerable as can be seen from Table 6.1. Nonetheless, it is still necessary to further decrease the rendering time in future work.

## 6.4 Summary

This chapter presented the simulation models of the scanning sensors and the GPU-based implementation of the simulation. The performance evaluation in terms of accuracy and computational cost was reported. It should be pointed out that the proposed simulation of scanning sensors and the related simulation framework are designed based on the work



(a) The zoomed out view.

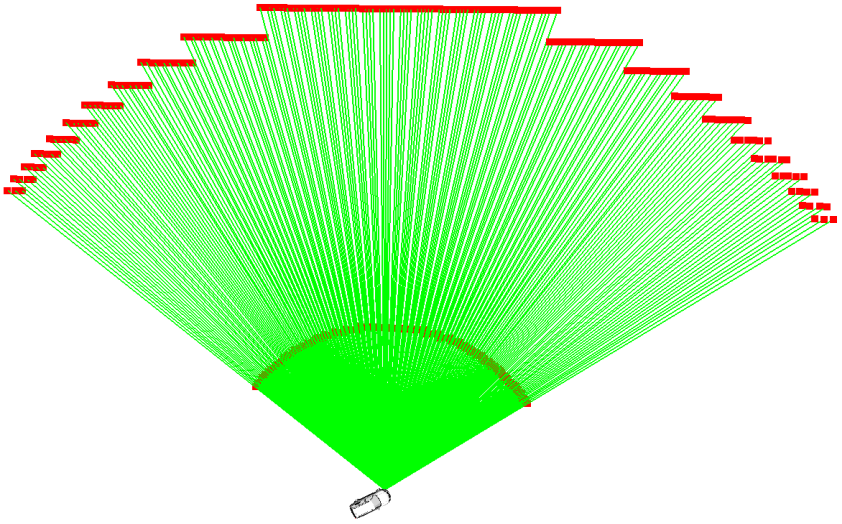


(b) The zoomed in view.

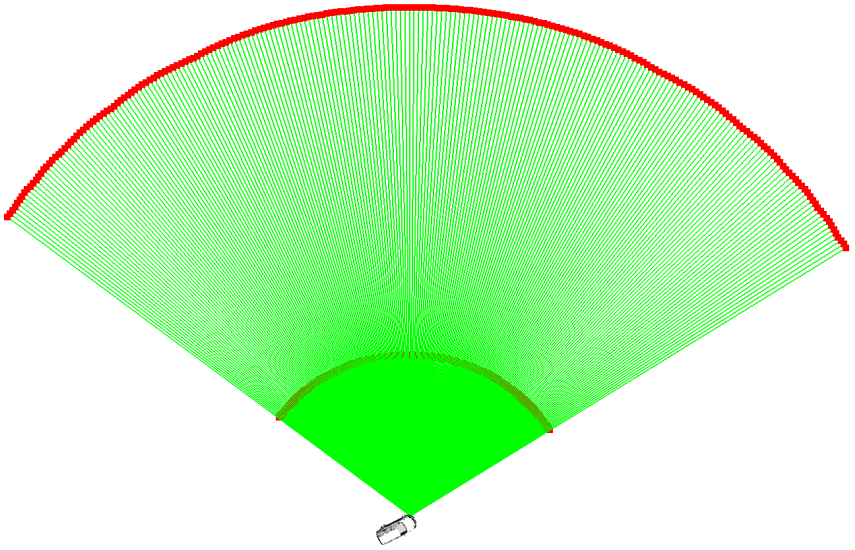
Figure 6.12: Using the pixels of the default to-be-rendered image to approximate the intersections of the actual laser beams and the near plane. The green points represent the pixels, while the red points display the intersections. The black points are those of the pixels that are nearest to the intersections and thus are chosen to be the approximations of the intersections. Note that the vertical interval between two neighbouring green points is so small that they are not distinguishable on the picture.

| Number of scene objects | Frame rate rendered by 6 IBEO sensors (The size of pixels for each sensor is $200 \times 400$) | Frame rate rendered by 6 IBEO sensors (The size of pixels for each sensor is $200 \times 2000$) |
|---|---|---|
| 0 | 34 | 30 |
| 1 | 33 | 24 |
| 2 | 31 | 18 |
| 3 | 25 | 13 |
| 4 | 20 | 11 |

Table 6.1: Computational costs for simulating IBEO sensors with different configurations. The rendering frame rate of the simulation environment without any simulated sensor is 34.



(a) The simulation result by directly using the pixels that are nearest to the intersections.



(b) The simulation result after the micro-scale correction is applied.

Figure 6.13: The effect of the micro-scale correction on the sensing result of two layers of laser beams that are reflected by the ground.

presented in [74]. The simulation approach employed in this thesis is similar in spirit to what is demonstrated in [68] and [69]. The proposed radar sensor simulation can generate the velocities of other traffic participants relative to the ego-vehicle which are one important output of the radar sensors mounted on autonomous vehicles. Regarding LiDAR, a macro-micro method was proposed to approximate the actual scanning pattern of the sensors and improve the accuracy of the simulated sensor data. Future work includes a validation of the simulated sensor data against the actual sensor data, enriching the physical phenomena considered in the simulation, adding proper noise models to the sensor simulation and decreasing the computational complexity of the rendering

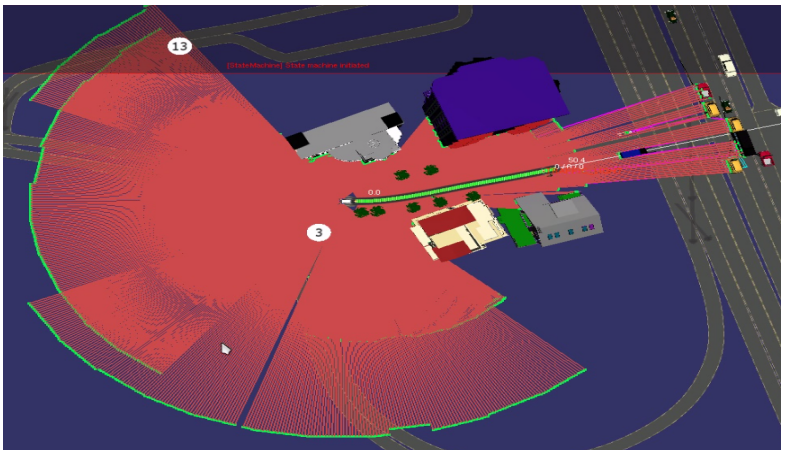


Figure 6.14: Simulation result for IBEO sensors. The detection range of the sensor is set to be 200 $m$.

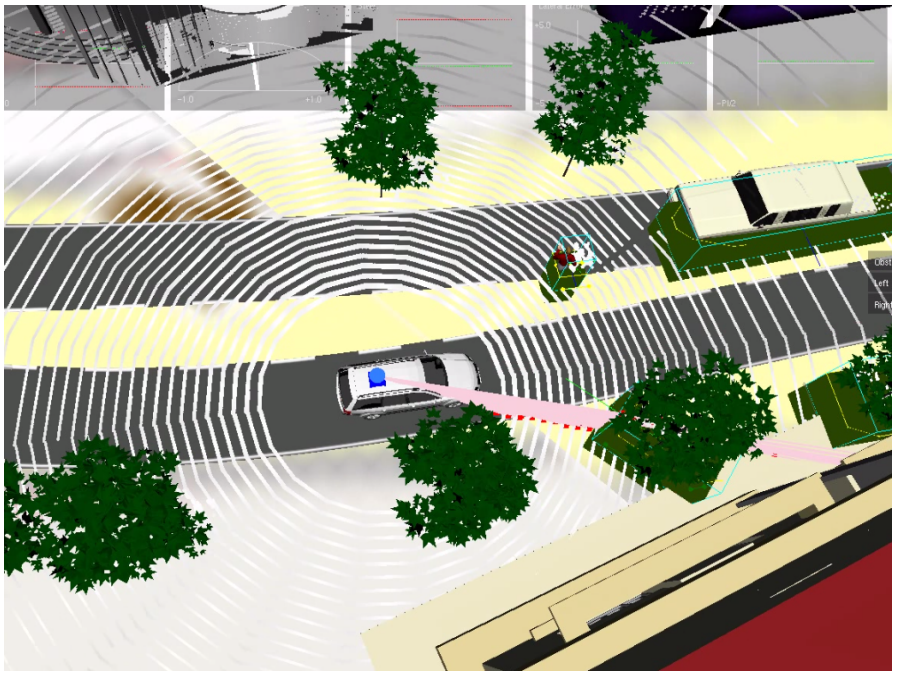


Figure 6.15: Simulation result for the Velodyne sensor. The white points forming concentric circles are the simulated sensor data interpreted by the obstacle detection module. The green boxes are the detection results of the obstacle detection and tracking modules. The pink lines demonstrate the rotating lasers.

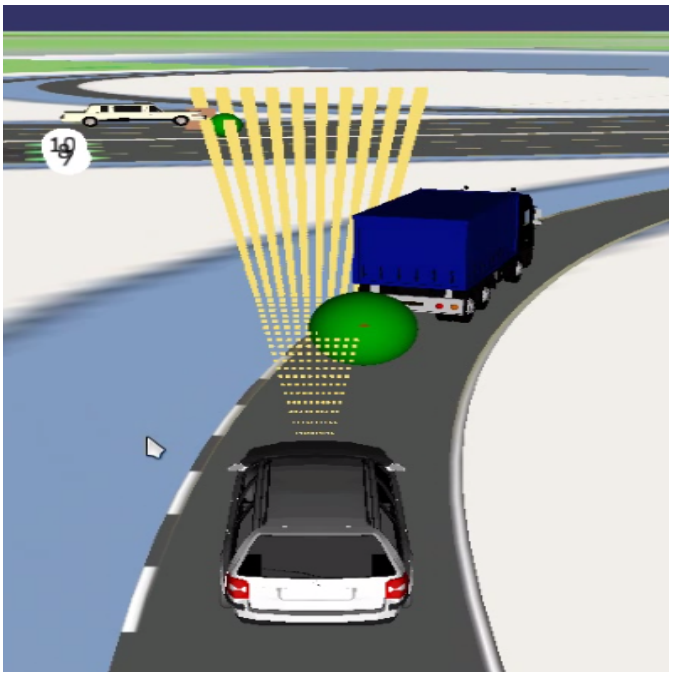


Figure 6.16: Simulation result for the radar sensor. The green balls are the detected reflectors. The HFOV of the simulated radar sensor is larger than its real-life counterpart for demonstration purpose.

of the sensors.

# Chapter 7

# Motion Planner Evaluation

In this chapter, the proposed motion planner is first examined according to the evaluation criteria illustrated in Chapter 1. After that, the capability of the proposed motion planner in handling time-critical traffic scenarios is reported. Then, the performance of the planner in driving the vehicle around road networks is displayed. Lastly, a comparison of the proposed planner and the planners presented in other works is given. It is noteworthy that all the presented experiments are carried out in a simulation environment, where the vehicle dynamics, the sensors and the traffic scenarios are all simulated. The software modules involved in the experiments, such as those for detecting and tracking obstacles, are directly adopted from the autonomous driving system of MIG. Figure 7.1 shows the software framework of the experimental platform. The hardware adopted here is the same as the one that is employed for the evaluation of the simulated scanning sensors (cf. Section 6.3). The configuration of the planner applied in the experiments follows Table 5.1 in general. The alterations of specific parameters required by the purposes of the experiments will be highlighted where necessary. As is mentioned earlier, there is no scenario reasoning module in the current implementation of the proposed planner. Correspondingly, the configuration parameters of the planner are adjusted manually on the fly when necessary. Besides, some scenario-related targets, such as target speed and location, are also set manually in runtime. For the corresponding videos, please refer to Appendix 1.

## 7.1 Criteria based Evaluation

In this section, the proposed planner is validated against the common criteria for evaluating motion planners, i.e., *completeness*, *feasibility*, *optimality* and *runtime* (cf. Subsection 1.3). In the evaluation, the following aspects are addressed for each criterion:

- The capability that the criterion expects the planner to have.
- The main factors that determine the performance of the motion planner in terms of the said capability.

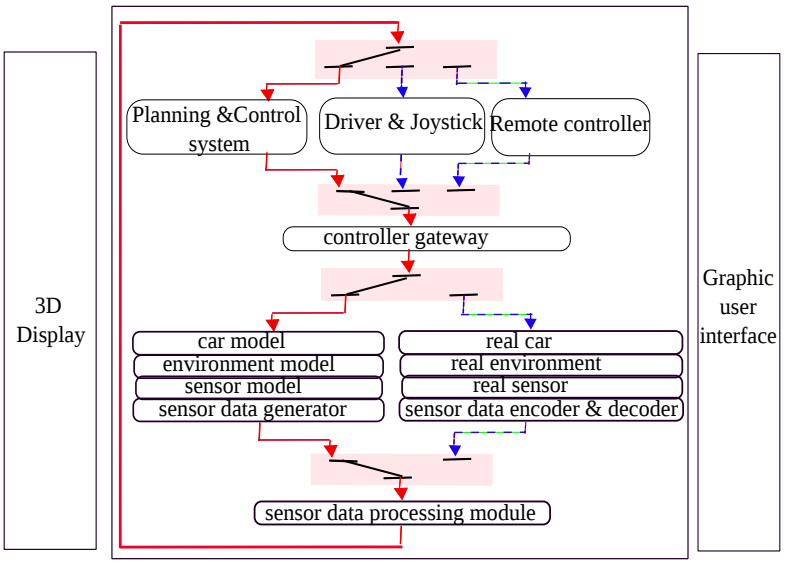



Figure 7.1: Software modules involved in the simulation environment where the experiments are conducted.

- Descriptions of the experiments conducted for the purpose of the evaluation and reports about the performance of the proposed planner demonstrated in those experiments.
- The properties that enable the proposed planner to achieve the satisfying performance, or the potential adaptations that can be made to improve the unsatisfying behaviour.

Some experiments address the potential impact of insufficient perception on the behaviour of the motion planner. The usefulness of the simulated scanning sensors can thus be demonstrated in the evaluation of the motion planner.

### 7.1.1 Optimality

The criterion of optimality can be further divided into *horizon optimality*, *scenario-dependant optimality* and *resolution optimality*. The quality of the proposed planner with respect to these three aspects of optimality is illustrated in the following.

#### 7.1.1.1 Horizon Optimality

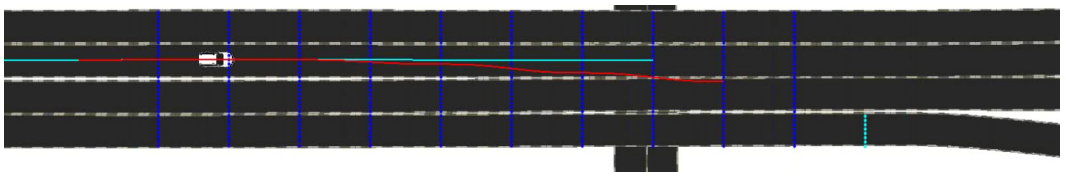
With limited planning horizon, the proposed motion planner can only achieve horizon optimality rather than global optimality. That is, the construction of the trajectories and the selection of the optimal one are based solely on the information available within the current planning horizon. Worse still, even such limited information cannot be fully utilized by the planner due to the insufficient perception of the vehicle. The criterion of horizon optimality is thus intended to assess the extent to which the best constraint-abiding trajectory satisfies a long-term goal. Such long-term goals can be, for example, a specific expectation of the speed of the vehicle, which is calculated based on the

investigations into the future. They can also embody the requirement of taking into account the complete information available within the planning horizon.

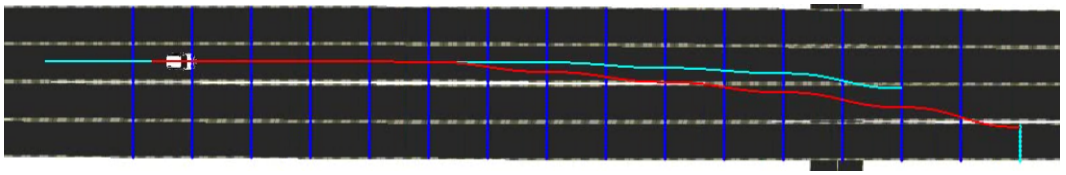
Figure 7.2 demonstrates a scenario where an extensive horizon can help the planner to recognize timely the necessity of deceleration for making a lane change within a short distance. Without any long-term information, the vehicle keeps running at high speed, which is an optimal behaviour from the perspective of the short planning horizon. By the time it realizes that it is necessary to make a lane change in order to drive onto the road that leads to the next checkpoint, there has been not enough room for the required manoeuvre. That is, the trajectories constructed in accordance with the limited connectivity pattern and the high speed are beyond the limited physical capability of the vehicle. The main constraint on the feasibility of the trajectory here is the maximum rate of change of curvature. As a result, the planner is left with no feasible plan, as is shown in Figure 7.2(a).

In contrast, a more extensive planning horizon adopted in the experiment shown in Figure 7.2(b) warns the planner of the necessity of a future lane change in time and thus makes it possible for the planner to plan a feasible trajectory with a timely deceleration. However, an extensive planning horizon will definitely deteriorate the computational efficiency of the planning. In this sense, a lightweight scenario reasoning module that can assign a long-term goal to the planner is a better choice than an extensive planning horizon. The issues revolving around scenario reasoning will be discussed in the next subsection along with the evaluation of scenario-dependent optimality.

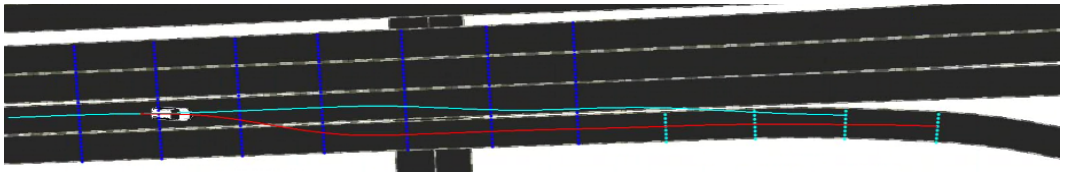
Now it is time to talk about the other kind of long-term goal, i.e., the requirement of considering the complete information within the planning horizon in the generation and selection of plans. Figure 7.3 and Figure 7.4 present different overtaking behaviours generated by the proposed planner in handling the same traffic scenario. In Figure 7.3, the static costs of the paths in terms of obstacles are evaluated based on the actual shape of the vehicle obstacle, and the ego-vehicle circumnavigates the static vehicle obstacle smoothly. In contrast, the overtaking manoeuvre generated based on the information provided by the simulated sensors shown in Figure 7.4 is not satisfying. When the ego-vehicle looks at the vehicle obstacle from behind, the perceived shape of the obstacle is much smaller compared to its actual shape. Consequently, the ego-vehicle takes a plan that is of high risk and even practically untraversable as demonstrated in Figure 7.4(a). When the vehicle discovered the "larger shape" of the obstacle, as is shown in Figure 7.4(b), it abandons the old plan (cyan) and generates a new trajectory (red) that is much safer. Finally, in Figure 7.4(c), the "looks" of the vehicle obstacle changes so drastically that all the trajectory edges from the ego-vehicle to the lattice turn out to be untraversable. Consequently, the planner fails to generate a plan. Such problems caused by insufficient perception can be mitigated by applying larger dilation around the obstacle and imposing heavier punishments when the ego-vehicle and the obstacle



(a) The planner fails to generate a feasible lane change manoeuvre at high speed. It has a short planning horizon of 100 $m$. A spatial node in the lattice is connected to up to $2 \times 9$ other spatial nodes.



(b) The planner succeeds in generating a feasible lane change manoeuvre with the help of an extensive planning horizon of 150 $m$. A spatial node in the lattice is connected to up to $2 \times 9$ other spatial nodes.

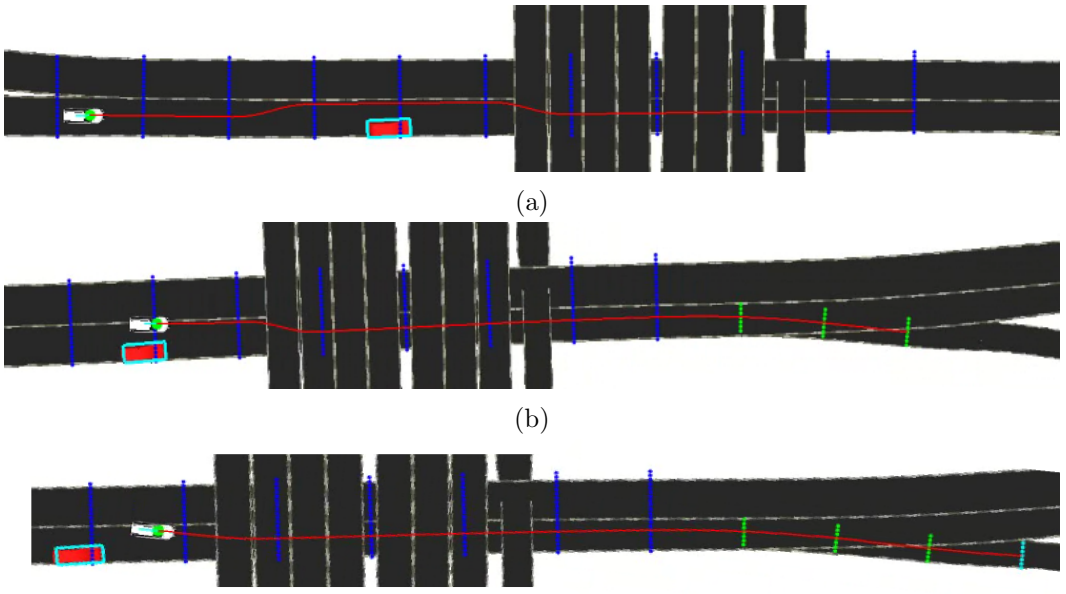


(c) The planner succeeds in generating a feasible lane change manoeuvre at high speed thanks to a richer connectivity pattern. Its planning horizon is 100 $m$. A spatial node in the lattice is connected to up to $3 \times 9$ nodes.

Figure 7.2: The lane change behaviours generated by the proposed planner with different planning distances and connectivity patterns. The red plan is younger than the cyan one. The several layers of points are the spatial samples of the planning horizon where the red plan is generated.

get too close. Nonetheless, a sufficient perception is desired in order to generate safer, more flexible and more consistent trajectories. It is noteworthy that only IBEO LiDAR sensors are employed on the autonomous vehicle in the simulation experiments. These sensors are installed at relatively low positions around the vehicle (cf. Figure 1.1) and have a very small VFOV, which makes it easy to obstruct their sights by a part of the whole obstacle. Should the Velodyne LiDAR sensor be mounted on the top of the vehicle ( cf. Figure 1.1), the perception of the vehicle would be much better.

There are also cases where scanning sensors cannot play a role in capturing the information of the obstacles. An example of such scenario is when a pedestrian appears suddenly from behind the static vehicles parked at the roadsides. In this case, a collision might be unavoidable. The perception system of the autonomous vehicle should be improved so that the information of the hidden traffic participants can also be taken into account in planning trajectories.



(a)

(b)

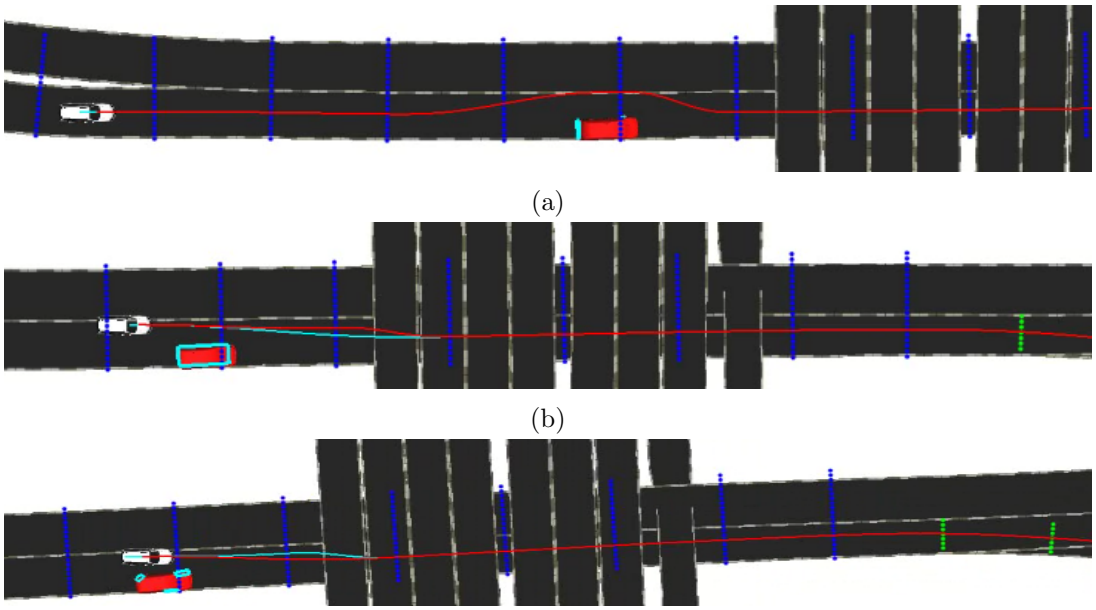
(c)

Figure 7.3: The overtaking manoeuvre generated based on sufficient information of the vehicle obstacle. The cyan box is the bounding box of the vehicle obstacle used for the construction of the cost maps. The several layers of points are the spatial samples of the planning horizon where the red plan is generated. The red plan is one planning cycle younger than the cyan plan.

#### 7.1.1.2 Scenario-dependant Optimality

The criterion of scenario-dependant optimality evaluates the planner's ability to adapt to the requirements of different traffic scenarios. Such adaptations can be an adjustment of the parameters of the cost functions, an alteration in the layout of the state lattice, etc. Accordingly, a scenario reasoning module is necessary which can determine the type of adaptation in order to satisfy the scenario-based requirements. Although the scenario reasoning module is absent in the current implementation of the proposed planner, it can be easily embedded in the planning architecture (cf. Section 5.3). The evaluation here focuses on the flexibility of the planner in response to given scenario-based instructions which are manually issued in the experiments.

Figure 7.5(a) shows the experiment where a target speed and a target location are assigned to the planner. To follow the instruction, the vehicle has to reach the target speed at the target location. As the current planner is not able to incorporate arbitrary locations into its set of lattice nodes, it can only direct the vehicle to reach a speed that is close to the target speed at the location that is near to the required target location. By activating the cost functions related to achieving a target speed, the planner generates a plan as demonstrated in Figure 7.6. As is mentioned in previous discussions about the criterion of horizon optimality, scenario-based adaptations can have the same effect as the application of an extensive horizon from the perspective of increasing the global



(a)

(b)

(c)

Figure 7.4: The unsuccessful overtaking manoeuvre generated based on insufficient knowledge about the vehicle obstacle which is provided by the simulated scanning sensors. The cyan box is the bounding box of the vehicle obstacle used for the construction of the cost maps.The cyan plan is older than the red one. Note that the vehicle obstacle gives different shapes when the ego-vehicle looks at it from different points of view. The planner fails to generate a traversable trajectory in the last picture as the "looks" of the vehicle obstacle changes so much that the predicted position of the ego-vehicle for the current planning horizon is untraversable itself. The several layers of points are the spatial samples of the planning horizon where the red plan is generated.

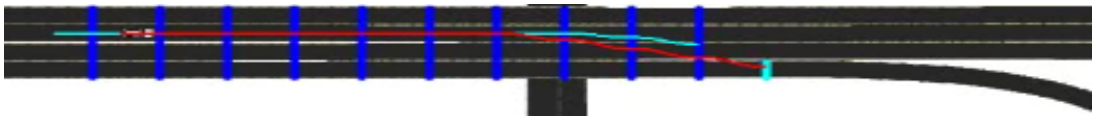
optimality of the optimal plan. In the example of the lane change manoeuvre at high speed demonstrated in Figure 7.2, a timely deceleration designated by the scenario reasoning module can make it possible for the planner to generate a feasible lane change manoeuvre later, as is shown in Figure 7.5(b).

It is noteworthy that it is the restriction of the maximum rate of change of curvature $\dot{\kappa}_{max}$ that makes all generated trajectories with high speed infeasible in the scenario shown in Figure 7.2(a). There is a relationship between the cubic spiral that represents the path edge and the rate of change of curvature along the trajectory edge, which is given as:

$$\dot{\kappa} = \frac{d\kappa(s)}{dt} = \frac{d\kappa(s)}{ds}\frac{ds}{dt} = \frac{d\kappa(s)}{ds}v \tag{7.1}$$

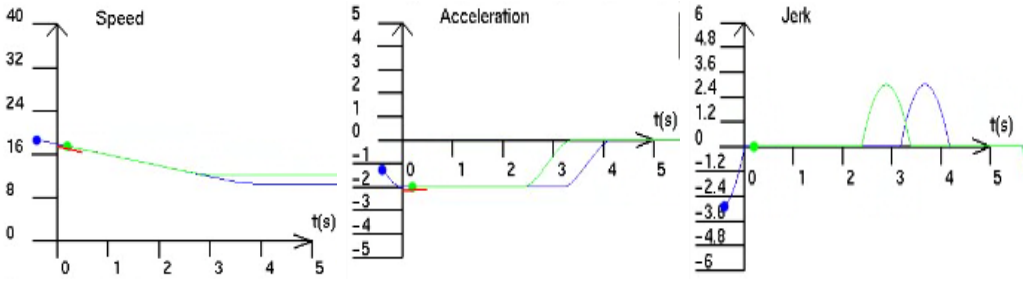
where $\kappa(s)$ refers to the expression of the curvature in terms of arc length (cf. Equation 3.7). Given a specific connectivity pattern and a straight road, the maximum $\frac{d\kappa(s)}{ds}$ can be easily derived. With the knowledge of the mapping between the connectivity pattern and the maximum $\frac{d\kappa(s)}{ds}$ corresponding to it, the planner can choose a connectivity pattern that can generate a trajectory with its maximum $\frac{d\kappa(s)}{ds}$ below $\frac{\dot{\kappa}_{max}}{v}$. In this way,

(a) A target speed that is expected to be reached by the vehicle at a specific target location is issued by the "manual" scenario reasoning module. The target speed is $10\,m/s$. The red point denotes the target location. The current speed of the vehicle is about $29\,m/s$.



(b) The lane change manoeuvre at low speed.

Figure 7.5: Deceleration to reach the target speed designated by a manual scenario reasoning module. The several layers of points are the spatial samples of the planning horizon where the red plan is generated. The red plan is one planning cycle younger than the cyan plan.

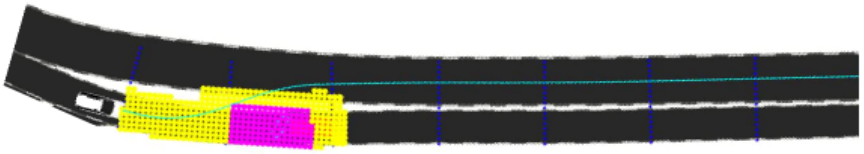



(a) (b) (c)

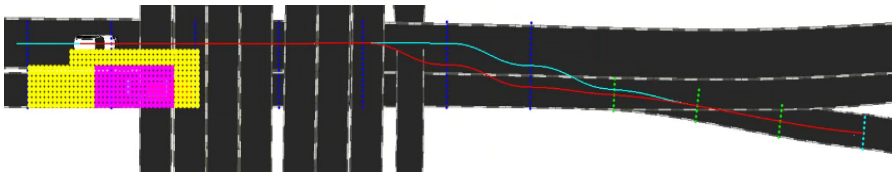
Figure 7.6: The plan in terms of speed, acceleration and jerk during the deceleration. The green plan is younger than the blue one. The short red curves record the five-second tracking result of the vehicle. The end of the planning horizon based on which the blue plan is generated has not reached the target location. The end of the planning horizon based on which the green plan is generated has passed the target location. The applied cost function for reaching a target speed only restricts the lattice nodes that have not passed the target location.

the planner is able to generate feasible trajectories as long as the required connectivity pattern can be realized in the lattice. Figure 7.2(c) shows the plan made possible by the connectivity pattern that can introduce the trajectory edges that satisfy the restriction imposed on the rate of change of the curvature.

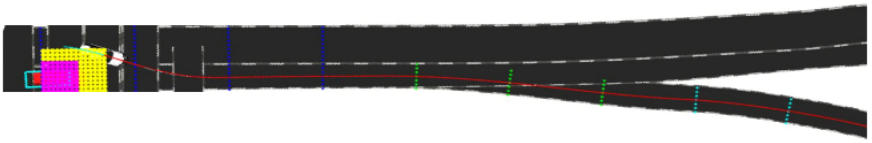
Another experiment where an adjustment of the weights of the cost functions is necessary is demonstrated in Figure 7.7(a). The neighbouring lane of the current travelling lane of the vehicle has oncoming traffic. As the criterion of time efficiency has a higher priority than the requirement of avoiding driving onto oncoming lanes, the ego-vehicle implements a risky manoeuvre that is intended to overtake the slow-moving vehicle obstacle via the oncoming lane. Such manoeuvre results in an even riskier behaviour later when it is necessary for the vehicle to return back to the original lane, as is shown in Figure 7.7(b) and Figure 7.7(c). The scenario reasoning module can predict the necessity



(a) A trajectory that makes the ego-vehicle take over the slow-moving vehicle in front of it via an oncoming lane.



(b) The ego-vehicle makes a risky lane change.



(c) The critical situation resulting from the risky lane change.

Figure 7.7: Risky trajectories generated by the planner without any far-sighted instruction from the scenario reasoning module. The yellow and magenta patches belong to the first frame of the cost map of dynamic obstacles. The yellow area is the high cost area, while the magenta patch refers to the fatal area. The actual shape of the vehicle obstacle, rather than the simulated sensor data, is used in this experiment. The several layers of points are the spatial samples of the planning horizon where the red plan is generated. The red plan is one planning cycle younger than the cyan plan.

of the second lane change based on the given road model; it can thus increase the cost for the vehicle to drive onto the oncoming lane. In this way, the aggressive trajectory shown in Figure 7.7(a) can be avoided.

#### 7.1.1.3 Resolution Optimality

The criterion of resolution optimality addresses the fact that the best constraint-abiding trajectory generated by the lattice-based planner is only local optimal. That is, the selected trajectory is only optimal from the perspective of the set of candidate trajectories that are constructed based on the given sampling pattern. The plan generated by the planner might not be the best one if all the possible trajectories in the continuous space are taken into account. The experiment shown in Figure 7.2(c) demonstrates the importance of dense sampling. It is also discussed in Section 4.4 that a rich portfolio of acceleration profiles will be beneficial in the generation of trajectories with a higher level of smoothness. One of the bottlenecks for increasing the density of the spatiotemporal sampling is the potentially intolerable computation time required by the construction of the dense state lattice.

### 7.1.2 Completeness

Like many other lattice based planners, the proposed planner has *resolution completeness*. That is, whenever there exists a solution in the continuous space, the planner will be able to find one provided that its resolution is fine enough; if there is no solution at all in the continuous space or when the solution in the continuous space cannot be generated by the planner due to its large resolution, the planner will report that no solution is found.

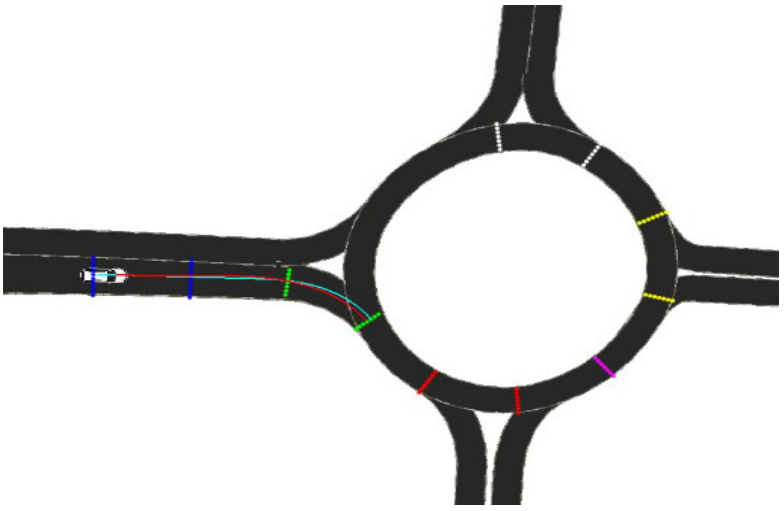
The experiment demonstrated in Figure 7.2 gives an example of the behaviour of the planner with resolution completeness. When the connectivity pattern results in a relatively small number of path edges outgoing from a spatial node, the planner reports that there is no feasible trajectory. When the number of outgoing path edges increases, the planner can return a feasible trajectory.

Figure 7.8 shows another experiment related to resolution completeness. In Figure 7.8(a), the planner is not able to find a valid path given a relatively large station interval. When the station interval gets smaller, as is shown in Figure 7.8(b), the planner can return a traversable trajectory. Such phenomenon occurs due to the requirements of the path edge (i.e.,curvature cubic polynomials) on its boundary conditions and length.

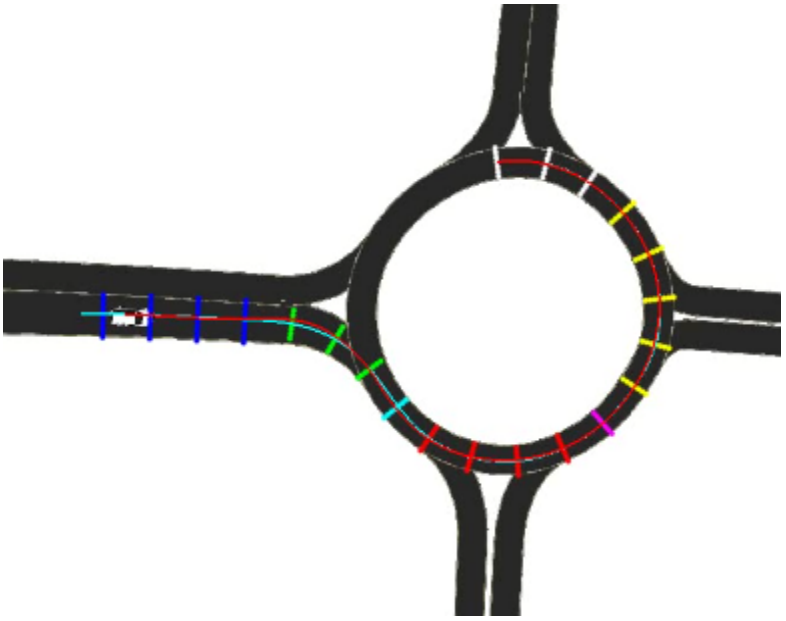
A planner with the property of *completeness* can demonstrate such performance: as long as there is a solution in the continuous space, the planner can return one; otherwise, the planner will terminate and report that no solution is available. There are in general two methods that can improve the condition of completeness of the proposed planner. One is to sample the continuous space as dense as possible; the other is to calculate explicitly a solution as long as there is one and to regard that solution as one candidate trajectory. The former will definitely lead to low computational efficiency, while the latter can be very complicated and may also be expensive. Nonetheless, it is worth the effort to approach *completeness* as much as possible for the sake of safety. Further improvements with regard to *completeness* are necessary.

### 7.1.3 Feasibility

The feasibility of the trajectory depends on the smoothness of the speed trajectory and the degree of continuity of the underlying path. The proposed planner can guarantee a higher level of feasibility by applying smooth acceleration profiles on the path edges in a consistent way. As for the feasibility of the path, it is pointed out in [25] and [23] that curvature polynomials with a higher order are required for the paths from the vehicle to the lattice in order to increase the consistency of trajectories generated in successive planning horizons. The curvature cubic polynomial applied currently can only ensure the continuity of the curvature at the joint of successive path edges, whereas the continuity of the first- and higher-order derivatives of the curvature with respect to the arclength



(a) No traversable and feasible trajectory can be found for the vehicle to drive onto the roundabout. The station interval is 10 $m$.



(b) Traversable and feasible trajectories can be constructed within the roundabout. The station interval is 5 $m$.

Figure 7.8: The performance of the proposed planner in dealing with the roundabout. The several layers of points are the spatial samples of the planning horizon where the red plan is generated. The red plan is one planning cycle younger than the cyan plan.

at the switching points cannot be guaranteed. This issue should be addressed in future work.

Another aspect of trajectory feasibility with regard to an on-road motion planner is the consistency of the plans generated from successive planning horizons. It can be observed from the experiments presented in this chapter that the plans are not always consistent. The main causes of the inconsistency are as follows:

- As the vehicle moves on, the planner receives updated information about the world model provided by the perception system. Subject to the parameters of related cost criteria, a small variation in the information might cause a considerable difference in the plans.

- The high level guidance sent from a behaviour module can also contribute to the inconsistent decisions of the planner.

- A precondition of consistency is that the best trajectory generated from the second planning cycle, or a part of it, can be found in the best trajectory produced from the first planning cycle. In this sense, even without the aforementioned external influences, the limited sampling and connectivity pattern of the planner can still lead to the inconsistency. The inconsistency issue with respect to the sampling of the spatial horizon has already been discussed in Chapter 5. With regard to the connectivity pattern, it is applied in the planner that the vehicle start pose can be connected to the spatial samples of a certain number of stations, and that each spatial sample is only allowed to connect to a fraction of spatial samples of one or two stations. As a result, the optimal trajectory generated in the following planning horizon might not be present in the previous horizon, which incurs inconsistency. Figure 7.9 illustrates this issue.

- It is also observed in the experiments that it can happen that the costs of some trajectories may turn out to be equal given no discretization artifact. In this case, slight variations in the cost maps or the sampling of the trajectories for evaluation will result in different judgements about those trajectory candidates.

The consistency of successive trajectories can be improved by increasing the sample number of the last plan for the next plan to follow and by raising the bonus for planning consistency. In this way, the planner will tend to choose the trajectory that is consistent with the last one as long as such a trajectory exists. This embodies a trade-off between *feasibility* and *optimality* of the plans. In the current implementation of the proposed planner, only the first two samples of the constructed trajectories are subject to such evaluation.

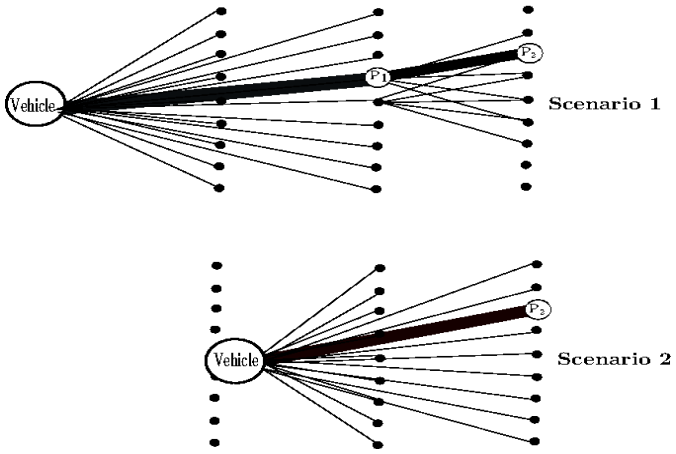



Figure 7.9: The inconsistency caused by the limited connectivity pattern. In the first scenario, the vehicle can only be connected to $P_2$ via $P_1$ due to the fact that the vehicle can only be connected to the first two stations in front of it. As the vehicle moves on, in the second scenario, it can be connected to $P_2$ directly. If the second connection turns out to be better than the first one, inconsistency between successive planning horizons occurs.

### 7.1.4 Runtime

In the CUDA-based implementation of the construction of the state lattice (cf. Section 5.2.2), each state lattice node initiates a thread executing the construction and evaluation of the trajectories that are originated from the said node. The construction of the state lattice is carried out in order of station. Consequently, the number of state lattice nodes of a station determines the amount of threads that are running in parallel for the construction of the trajectories outgoing from this station; the computational cost for the execution of each thread depends on the number of trajectory edges outgoing from each lattice node. Let $n_l, n_v, n_t, n_\alpha, n_s$ denote the number of discrete latitudes, speeds, times, accelerations and stations respectively. Let $n_\rho$ refer to the number of outgoing path edges of a lattice node. The maximum number of threads initiated by each station is given as $n_l n_v n_t n_\alpha$, and the maximum amount of trajectory edges that are constructed within one thread is calculated as $n_\alpha n_\rho$. In practice, it might happen that some lattice nodes are not represented. That is, no trajectory edge ends at those lattice nodes. As a result, the threads initiated by those lattice nodes will stop being executed very soon. It is left for the GPU to schedule the busy or vacant threads executed in the multiprocessors. The way in which the GPU schedules the threads, together with the number and layout of the multiprocessors, determines to a large extent the parallelism of the algorithm for the construction of the state lattice. Let $N_P$ refer to the number of threads that are practically executed in parallel. The computational complexity of the CUDA-based implementation of the state lattice construction is given as:

$$\mathcal{O}(\frac{n_l n_v n_t n_\rho n_\alpha^2 n_s}{N_P}). \tag{7.2}$$

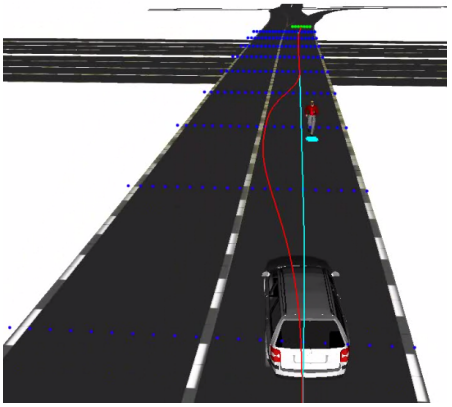
Given the configuration of the state lattice shown in Table 5.1, the time taken by each phase of the planning cycle is displayed in Table 5.2. In future work, the CUDA-based implementation can be further optimized to make full use of the parallel computing architecture of the GPU.

## 7.2 Planner Performance in Time Critical Scenarios

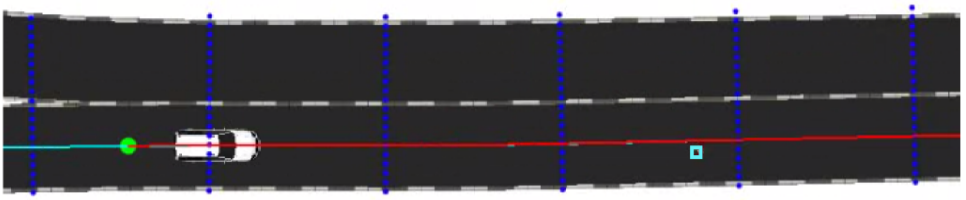
This section reports the performance of the proposed planner in handling time-critical scenarios. Two scenarios are chosen for the experiments. One requires the vehicle to perform emergency evasive manoeuvres where making a panic stop cannot avoid the collision. In the other scenario, the vehicle needs to merge into traffic, which requires careful timing and speed control.

### EMERGENCY EVASIVE MANOEUVRE

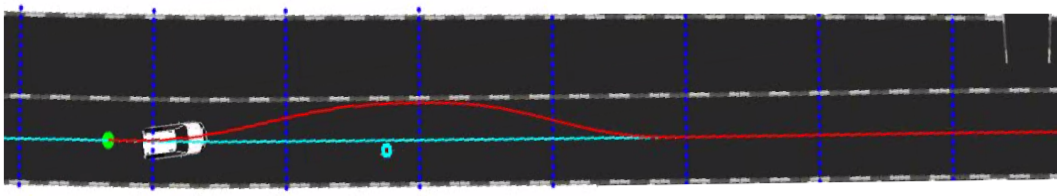
In the traffic scenario shown in Figure 7.10(a) and Figure 7.10(b), the vehicle is accelerating at the hardest possible acceleration, i.e., $2\,m/s^2$, as the configuration of the cost functions gives the highest priority to the criterion of time efficiency. At the moment when the speed of the vehicle reaches $15\ m/s$, a pedestrian appears suddenly in the center of the road in front of the vehicle. The distance between the vehicle and the pedestrian is $25\,m$. In real life, such scenario can happen because some other obstacles blocked part of the view of the vehicle so that it could not see the pedestrian when he was still close to the roadside. According to the equation $s = v^2/(2\alpha)$, a distance of $28\,m$ is required for the vehicle to come to a stop from running at $15\,m/s$ when the hardest possible deceleration is $-4\ m/s^2$. As the actual distance is $25\ m$, it is not enough for the vehicle to implement a panic stop in order to avoid a collision with the pedestrian. Actually, the distance between the pedestrian and the vehicle from the perspective of the planner is even smaller than $25\ m$ as the planner needs to simulate forward the vehicle dynamics system for a specific duration to compensate for the planning latency. This phenomenon can be observed by comparing the distances of the pedestrian (the cyan box) relative to the vehicle (in Figure 7.10(b)) and to the green point (in Figure 7.10(c)). Consequently, the planner generates an evasive manoeuvre composed of a double lane change as demonstrated in Figure 7.10(c), Figure 7.10(d) and Figure 7.10(e). The first lane change is for evading the obstacle, while the second one is for returning back to the original travelling lane. Figure 7.11 displays the trajectories of curvature, speed, acceleration and jerk of the double lane change. In this way, the vehicle avoids potential collisions with the pedestrian. It should be pointed out that, in this experiment, the pedestrian is assumed to stand still in the road after his appearance.



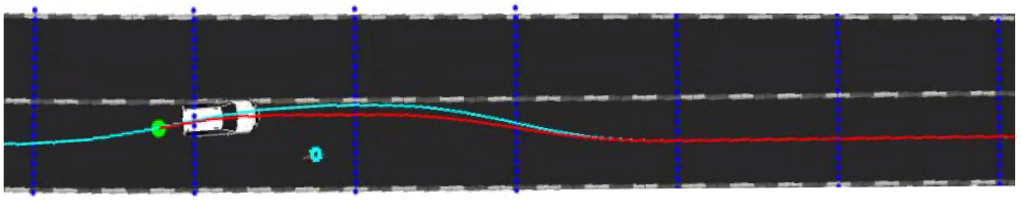
(a) A pedestrian leaps into the view of the vehicle suddenly (a snapshot from the perspective of the vehicle).



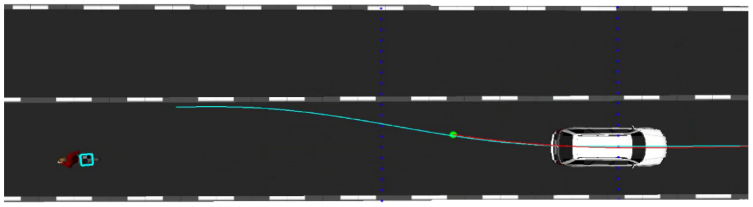
(b) The trajectories that are currently executed by the vehicle at the moment when the pedestrian is located.



(c) The trajectory generated for the vehicle to swerve to avoid the pedestrian.



(d) Another trajectory generated when the vehicle executes the evasive manoeuvre.



(e) The vehicle at the end of the evasive manoeuvre.

Figure 7.10: The trajectories generated by the proposed planner in handling an emergency scenario. In each picture, the red line refers to the plan generated by the last planning cycle and is currently executed by the vehicle. The cyan line is the plan that is a planning cycle older than the red one. The green point indicates the predicted starting position of the vehicle for the red plan. The cyan box is the bounding box of the pedestrian used in the construction of the cost map of obstacles. The several layers of points are the spatial samples of the planning horizon where the red plan is generated.

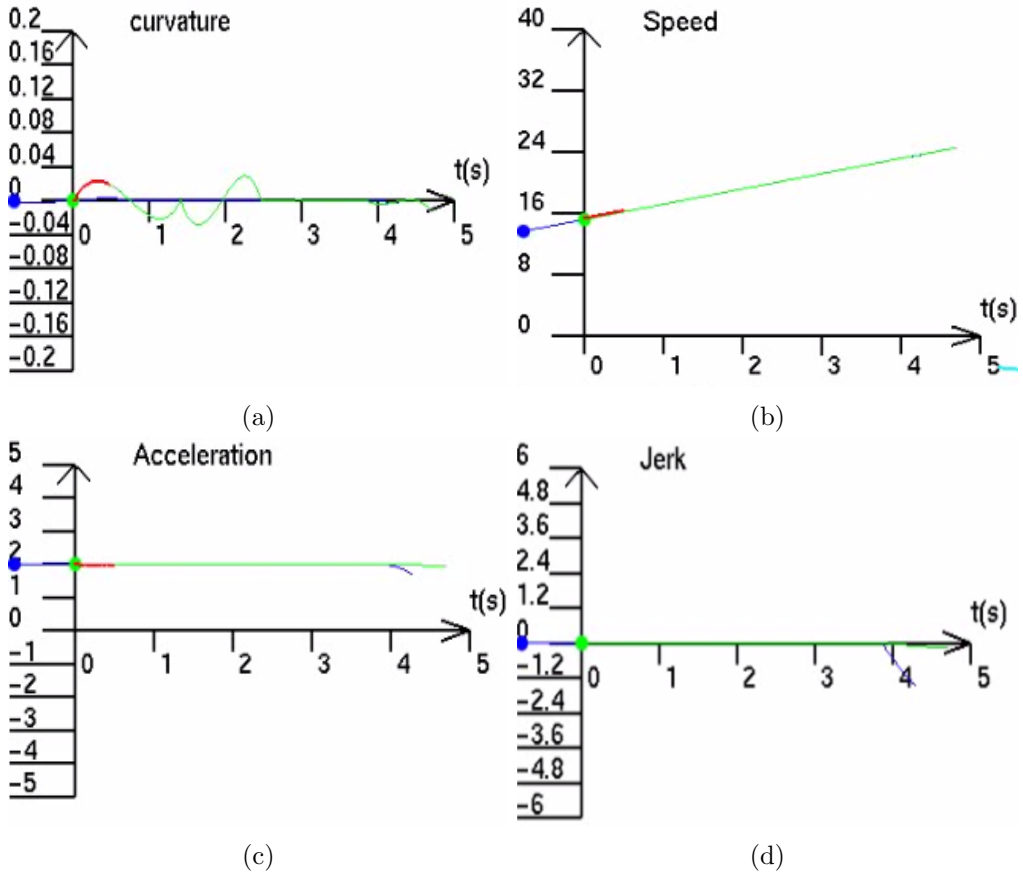



Figure 7.11: The trajectories of curvature, speed, acceleration and jerk for the double lane change. The green and blue curves correspond to the red and cyan plans in Figure 7.10(c), respectively. The short red curves record the five-second tracking result of the vehicle.

### MERGING INTO MOVING TRAFFIC

Merging is among the most frequently executed complex driving manoeuvres. In the experiments shown in Figure 7.12, three vehicles are running in a queue along the lane that is next to the acceleration lane. Their speeds are $30m/s$, and successive vehicles keep a three-second interval. The ego-vehicle runs at 10 $m/s$ when it is about to accelerate on the acceleration lane in order to merge into the traffic on the lane to its right. If no lane is designated to the planner onto which the ego-vehicle should travel, the trajectory that is displayed in Figure 7.13 is generated by the planner. Since the criterion of time efficiency is given top priority, the ego-vehicle prefers not to stay in the traffic into which it merges; it drives onto the next neighbouring lane so that it can run at a speed higher than that of the other vehicles. When the lane on which the other vehicles are running is assigned to the planner through giving a huge bonus to the candidate trajectories ending at the said lane, the vehicle ends up performing what is shown in Figure 7.14.

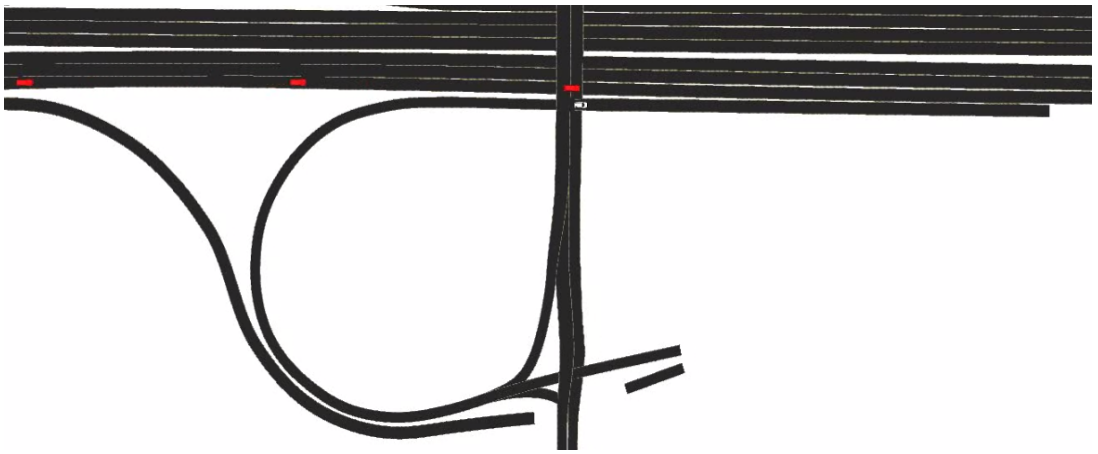


Figure 7.12: An example scenario where the ego-vehicle is required to merge into the traffic on the neighbouring lane.

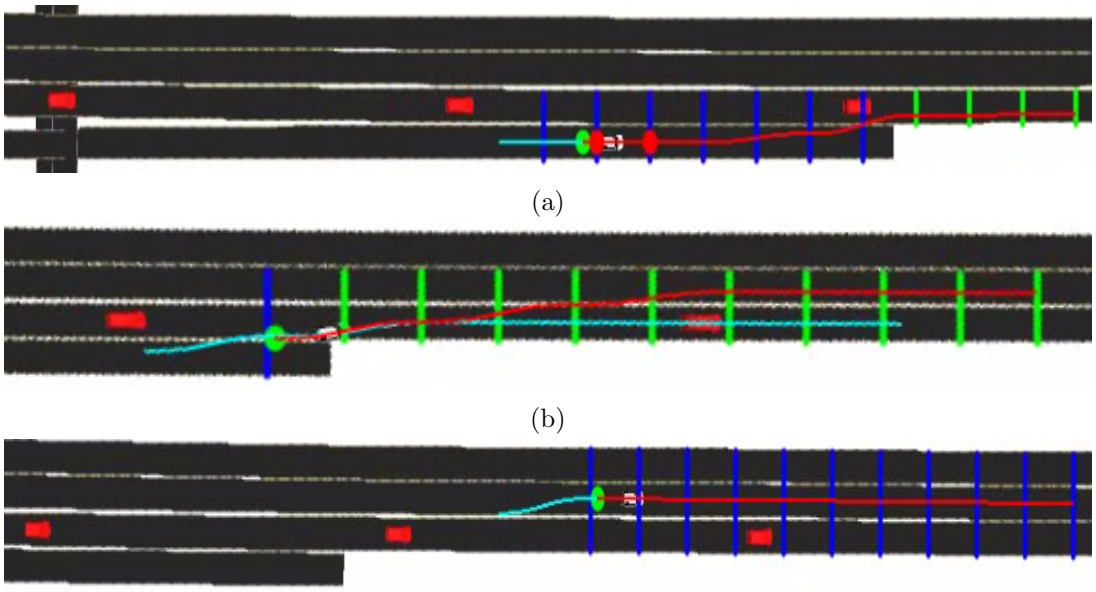
Figure 7.15 displays the trajectories of curvature, speed, acceleration and jerk of the merging manoeuvre demonstrated in Figure 7.14(a). It can be concluded from these experiments:

- The proposed planner can generate reasonable merging behaviours under proper guidance. However, the plans generated in successive planning horizons might not be consistent as can be seen in Figure 7.14(b). There are potentially several reasons for that (cf. Subsection 7.1.3). The cause of the planning inconsistency demonstrated in Figure 7.14(b) is that the new plan (red) cannot be constructed in the previous planning horizon due to the limited connectivity pattern.
- A scenario reasoning module is necessary to regulate the behaviour of the planner.
- In real-life traffic, the vehicle behind the ego-vehicle will adjust its speed due to the merging manoeuvre of the ego-vehicle. Future work regarding simulation should take into account potential interactions between the traffic participants.

## 7.3 Planner Performance in Road Network Experiments

The planner is tested in three road networks as demonstrated in Figure 7.16, Figure 7.17 and Figure 7.18. The main purposes of these experiments are as follows:

- To check the general applicability of the proposed strategy for specifying the spatial horizon.
- To examine the consistency and feasibility of the generated trajectories.
- To test the capability of the planner in navigating the vehicle among moving traffic in road networks.

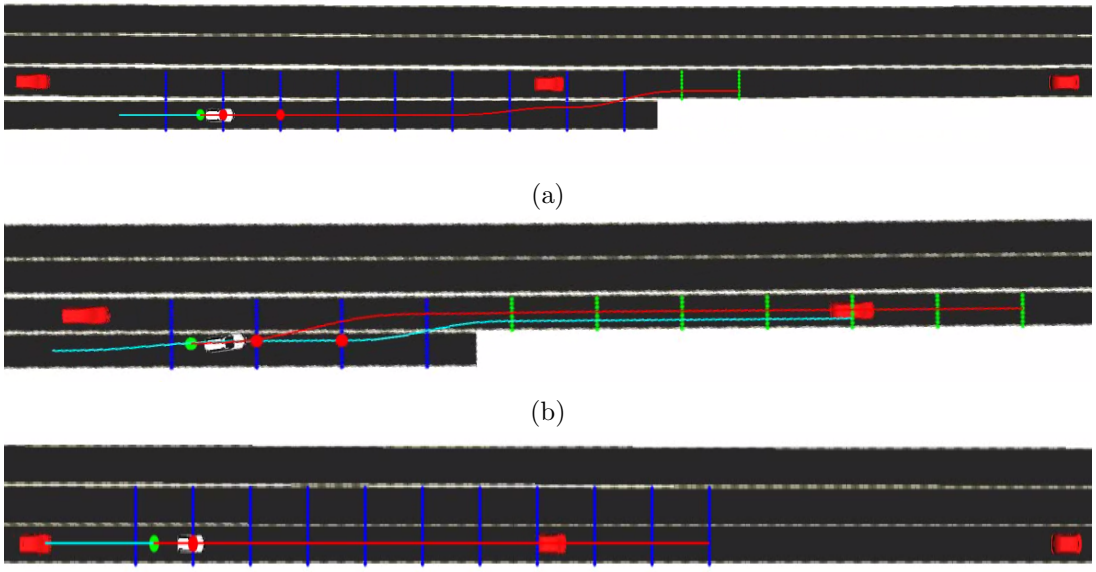


(a)

(b)

(c)

Figure 7.13: The merging behaviour when there is no designated lane for the planner, and the criterion of time efficiency is given top priority. See Figure 7.10 for the meanings of the points and curves.



(a)

(b)

(c)

Figure 7.14: The merging behaviour when the neighbouring lane of the acceleration lane is designated to the planner as the target lane. See Figure 7.10 for the meanings of the points and curves.

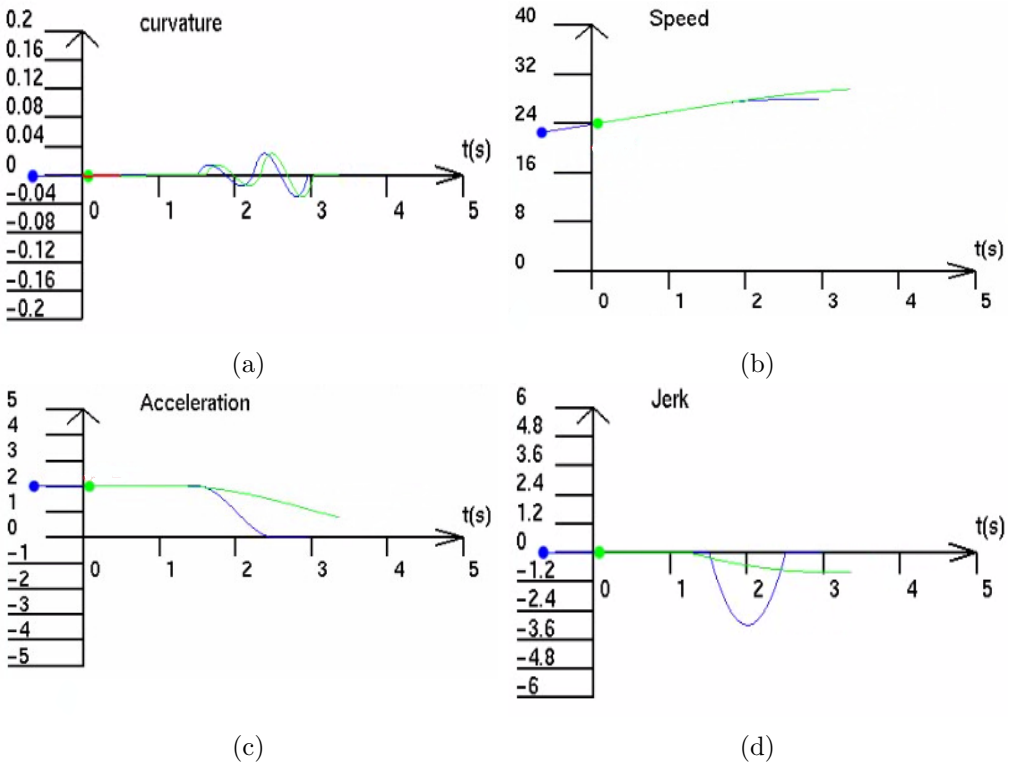



Figure 7.15: The trajectories of curvature, speed, acceleration and jerk corresponding to the plans shown in Figure 7.14(a). The green and blue curves correspond to the red and cyan plans in Figure 7.14(a), respectively.

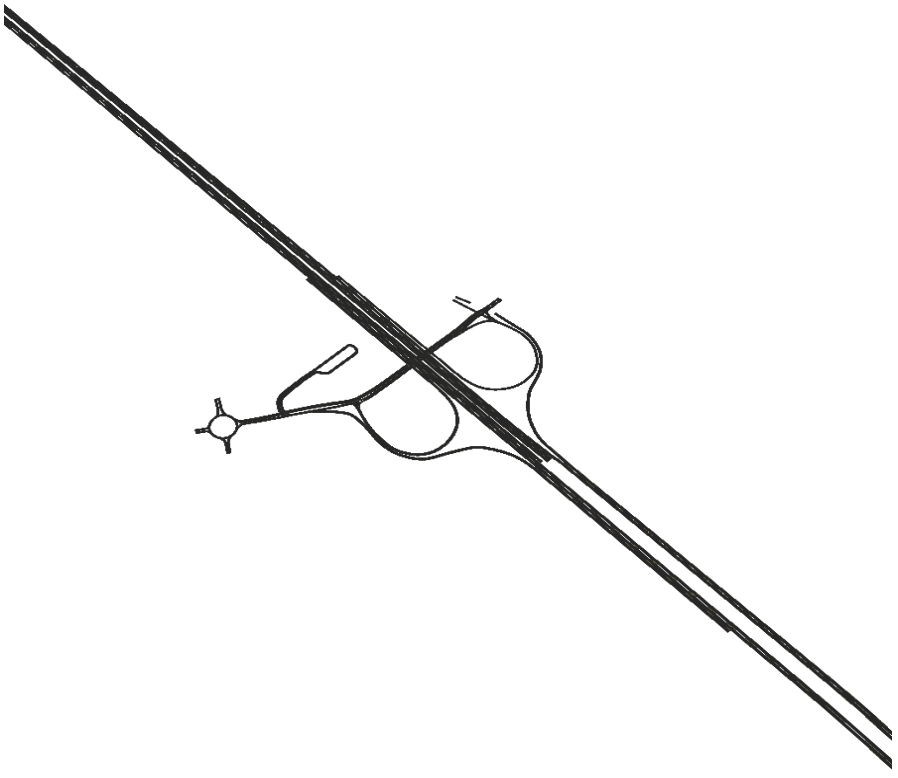


Figure 7.16: Road network *Avus*.

Figure 7.17: Road network *Dahlem*.

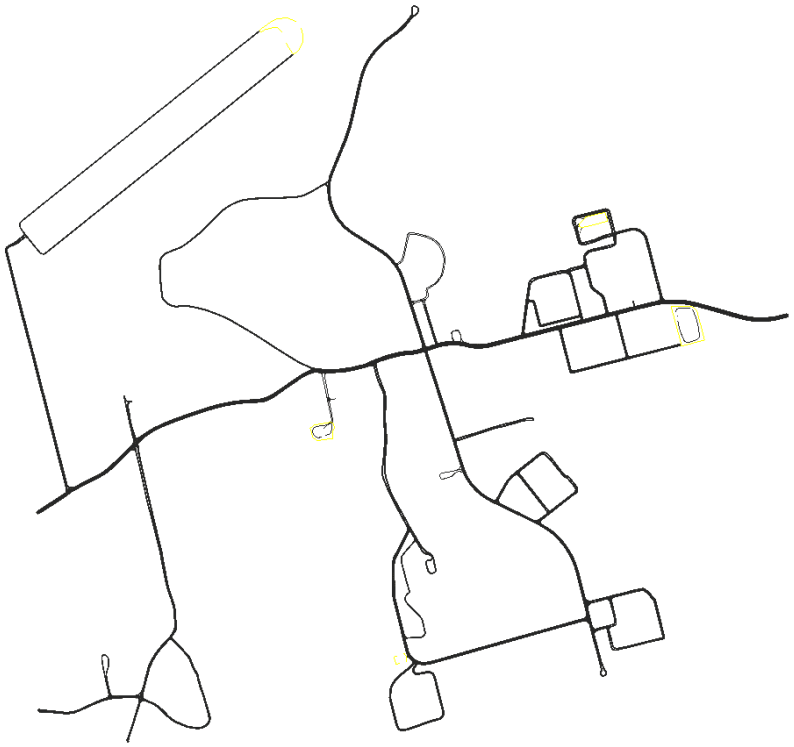


Figure 7.18: Road network *Freiburg*.

It is noteworthy that the experiments are only carried out in a simulation environment, where the road networks are modelled based on the real ones. Only with sufficient traffic perception and prediction, a sophisticated scenario reasoning module and a capable controller can the experiments be performed in real road networks. Chapter 8 will discuss more about that. The configuration parameters and scenario-related targets that can be adjusted or set in runtime include:

- Station interval. A $10m$ interval is applied for most driving roads; smaller intervals (e.g., $5\,m$) are necessary for road segments with sharp curves.
- The weight of the cost item for promoting time efficiency. Sometimes, the vehicle can be "trapped" into running at a rather small speed due to the tendency of the planner to avoid punishments incurred by a high rate of change of curvature or a large lateral acceleration. Further, if the bonus for following the last plan is set too high, the vehicle would keep that low speed for a long time until some other factors challenge the conservative movement. In such circumstance, it is necessary to increase the cost for wasting time so that the vehicle can move faster. Besides, it can happen that the cost for approaching other travelling vehicles is so high that the ego-vehicle keeps following the vehicle running slowly ahead of it. In this case, a higher penalty imposed on slow driving can encourage the ego-vehicle to overtake the slow vehicle and go on driving at a higher speed.
- Target speed and distance. As is mentioned previously, the planning horizon is short in order to achieve computational efficiency. Consequently, it is necessary to inform the vehicle of the necessity to slow down so that it can, for example, turn around a sharp curve successfully.

Equipped with the scenario-related guidances listed above, the planner can drive the vehicle smoothly and safely around the road network most of the time. Still, there are cases where the planner generates inconsistent plans or fails to generate a feasible and traversable trajectory. The main reasons for the inconsistency have already been illustrated in Subsection 7.1.3. The causes that lead to the failures discovered so far are listed below:

- Untimely or improper manual scenario-dependant guidance. For example, the vehicle cannot make a sharp turn at a high speed due to the limitations of its physical capabilities, such as the restricted deceleration, rate of change of curvature and lateral acceleration. Accordingly, if a target speed is not issued timely or properly to tell the vehicle to slow down, there will be no physical constraint-abiding trajectory for the planner to construct and choose.

- "Infinite cost" effect. Sometimes the planner may generate a trajectory that lies very close to the infinite area of certain cost items in order to minimize the resultant summed cost. In this case, a slight change in the perceived world model or the discretized cost maps or a small tracking error may cause the vehicle start state or the originally valid trajectory to be trapped into an "infinite cost' area in the subsequent planning horizon, which can render all trajectories from the vehicle infeasible or untraversable.

- Challenging road structures. Due to the physical constraints of the vehicle and the intrinsically limited sampling and connectivity pattern of the planner, the planner may fail to find a valid trajectory through a sharp road curve with a limited width.

With the existing problems related to spatial horizon specification and sampling ignored (cf. Section 5.3), the proposed strategy for specifying the spatial horizon works consistently in the three road networks.

The jerk levels of the trajectories traced out by the vehicle equipped with the proposed motion planner in the road network experiments are displayed in Table 7.1 and Table 7.2. The *distance jerk level* in Table 7.2 is calculated according to Equation 4.3 where $x$ refers to the travel distance of the longitudinal movement of the vehicle. The *heading jerk level* is computed based on:

$$J(\kappa(t)) = \frac{1}{2}\int_{t_0}^{t_f} \dot{\kappa}(t)^2 dt \tag{7.3}$$

where $\kappa$ refers to the curvature of the trajectory. It is noteworthy that in practice the discretized state samples of the trajectory are applied in the calculation as is shown in Equation 7.4:

$$\begin{aligned} J(x(t)) &= \tfrac{1}{2}\sum_{k=0}^{N-1}(\tfrac{\alpha_{t_{k+1}}-\alpha_{t_k}}{t_{k+1}-t_k})^2(t_{k+1}-t_k) \\ J(\kappa(t)) &= \tfrac{1}{2}\sum_{k=0}^{N-1}(\tfrac{\kappa_{t_{k+1}}-\kappa_{t_k}}{t_{k+1}-t_k})^2(t_{k+1}-t_k) \end{aligned} \tag{7.4}$$

where $\alpha$ refers to *acceleration*. For comparison purposes, the performance of the path planner (cf. [1] [88]) that is currently deployed on MIG is also shown in Table 7.2. Several issues are noteworthy for comprehending the displayed data:

- The distance jerk level calculated according to the acceleration samples provided by the simulator reflects a comprehensive effect of the planner, the controller and the simulated vehicle model. In comparison, the distance jerk level computed based on the jerks expected from the controller indicates only the performance of the planner. The controller for the motion planner extracts the expected jerks from the plans, while the controller for the path planner issues the throttle actuation command based on the current speed and the desired speed. The desired speed is

calculated by the controller for the path planner according to some considerations, such as the speed limit of the current road and the physical limits of the vehicle. As a result, there is no explicitly intended jerk from the path planner and its controller, which makes it impossible to calculate the expected-jerk based jerk level of the path planner.

- As can be seen from Table 7.2, there are huge discrepancies between the expected-jerk based jerk level and the simulation-based jerk level. That is due to the failure of the controller in guiding the vehicle according to the expected acceleration. One phenomenon that contributes a lot to the discrepancy is shown in Figure 7.19, where the acceleration has to oscillate about the zero axis in order for the vehicle to maintain a constant speed.

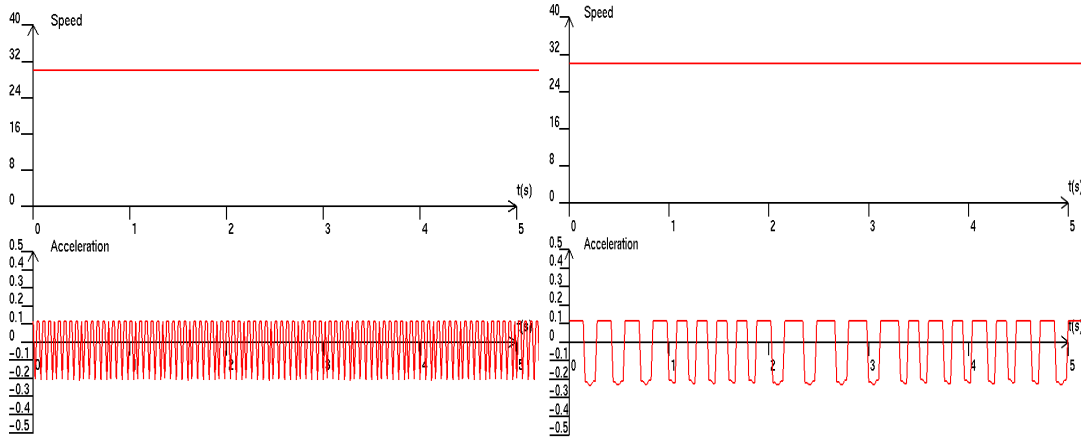
- The controller for the path planner issues actuation commands at an interval of $0.04 seconds$, while the controller for the motion planner sends actuation commands every 0.001 *seconds*. The simulator updates the vehicle states every 0.02 *seconds*. The effect resulting from the arrangement of the update periods can be observed through the oscillation periods of the acceleration shown in Figure 7.19. Consequently, the calculation of the jerk level which is based on discrete state samples must be affected by the update periods.

- According to the two issues mentioned above concerning the effectiveness of the calculation method of the distance jerk levels, it is actually hard to compare the distance jerk levels of the trajectories designed by the motion planner and the path planner plus its controller based on the jerk levels calculated in that way. Consequently, the data related to distance jerk levels displayed in Table 7.2 cannot be used to compare the two planners; instead, they are demonstrated to give a hint as to how large the distance jerk levels can be and to compare the real performance of the vehicle with what the motion planner expects.

- As the model of the vehicle lateral movement is rather ideal in the simulation, the heading jerk level calculated based on the discretized curvatures provided by the simulator can reflect the heading performance expected from both planners. However, it should always be kept in mind that the amounts of heading jerk levels calculated with the method presented in this thesis are still subject to the update periods of the controllers.

It can be concluded from Table 7.2 that:

- The heading jerk level of the path planner is always lower than that of the motion planner. Although the quantities of the discrepancies are questionable according to what is discussed above, their existence reflects what can be observed in the




(a) The acceleration oscillation caused by the controller for the motion planner.

(b) The acceleration oscillation caused by the controller for the path planner.

Figure 7.19: The acceleration oscillation that appears when the controller tries to keep the vehicle at a constant speed.

experiments; that is, the path planner tends to be better than the motion planner in terms of heading jerk levels. That is because the planning horizon of the path planner is four or five times more extensive than that of the motion planner. Consequently, the path planner tends to distribute the change of heading over a longer distance, which leads to a smaller heading jerk. Besides, the limited connectivity pattern of the motion planner also prevents it from achieving lower heading jerk level.

- Due to the factors that can affect the calculation result of the distance jerk levels mentioned previously, it is hard to tell exactly from the calculated results which planner performs better in terms of distance jerk level. Nonetheless, the jerk level of the motion planner can be further decreased when the jerk level criterion is introduced to the evaluation of the trajectories. Besides, once the restrictions of the computational complexity gets further relaxed, the planning horizon of the motion planner can be further extended, which will enhance its performance in terms of jerk levels. In contrast, the path planner does not have the potentials of the motion planner illustrated above because it can only generate plans in the spatial space.

- The substantial discrepancy between the simulation-based and the expected-jerk based jerk levels conveys a message that only with a capable controller can the full potential of the motion planner be explored.

Figure 7.20 illustrates an example of the reactions of the planner to the surrounding traffic participants. The vehicle can respond to the moving traffic reasonably given proper guidance from the scenario reasoning module. Without adjustment of the weight

of the cost criterion for promoting time efficiency, the behaviour of the vehicle might tend to be unreasonably conservative or aggressive.

It is noteworthy that the simulated traffic participants cannot interact with the ego-vehicle, and the planner treats all the traffic participants, either behind or in front of the ego-vehicle, equally. It should also be pointed out that the future routes of the vehicle obstacles are known *a priori* from the perspective of the planner in the experiments. That is, the planner predicts the future trajectory of the vehicle obstacle according to the same rule applied in the traffic simulation. For example, the simulated vehicle obstacle always keeps at the center of the travelling lane; when confronted with road branches, it will travel onto the first branch that is returned by the branch search function. In this sense, only the speed, the shape and the initial position of the obstacles at the beginning of each planning cycle keep in accordance with the perception result of the simulated scanning sensors; the prediction of future route is not based on the sensed heading direction of the vehicle obstacle. Such relatively ideal prediction strategy has to be taken because the currently available obstacle prediction module can only predict the future route of the obstacle based on the assumption of a straight line movement. That is, it does not take into account the shape of the road where the vehicle obstacle is running. Such prediction method is rather insufficient for supporting a motion planner whose planning result relies largely on the prediction of the surrounding obstacles. A good prediction of the future route of the obstacle is necessary, but devising such a module that can fulfil the task is beyond the scope of this work. Consequently, the aforementioned *a priori* prediction strategy is designed and utilized in the experiments.

In Figure 7.20, the sensed obstacles (cyan boxes) are the looks of the real obstacles from the perspective of the planner at the assumed starting time ( $t_{red-start-assumed}$) of the red plan. Let the time when the simulated sensors located the obstacles be denoted as $t_{scan-begin}$, and it takes $\delta t_{scan-generate}$ for the simulated sensors to generate the scanning information and send it to the obstacles processing modules. Let $t_{scan-end}$ denote the time when the sensor data is generated. It follows that $t_{scan-end} = t_{scan-begin} + \delta t_{scan-generate}$. In the current implementation of sensor simulation, $\delta t_{scan-generate}$ is ignored. That is, it is assumed that $t_{scan-begin}$ is equal to $t_{scan-end}$. As $t_{scan-end}$ can be obtained directly, it is assumed that the obstacles were located by the sensors at $t_{scan-end}$ rather than the real time $t_{scan-begin}$. At the beginning of the planning cycle, the planner predicts the locations of the sensed obstacles at time $t_{red-start-assumed}$ and outputs the predicted obstacles to the display module. At the time when the snapshots of the figures in Figure 7.20 are made, the red plan is already ready, and the planner is deliberating about the next plan. Let the difference between the time of the snapshot and the assumed starting time of the red plan be denoted as $\delta t_{passed-from-red-start-assumed}$. The displacements $\delta s_{box}$ of the displayed cyan boxes from where the real obstacles were at $t_{scan-begin}$ are:

| Track | Vehicle obstacles | Speed limit (m/s) | Travel distance (m) | Duration (s) | Examples |
|---|---|---|---|---|---|
| Avus-0 (Figure 7.16) | 33 | 30 | 18963 | 1312 | Figure 7.20 |
| Avus-1 (Figure 7.16) | 0 | 30 | 19473 | 1040 | Figure 7.21 |
| Avus-2 (Figure 7.16) | 0 | 50 | 19512 | 882 | Similar to Figure 7.21 |
| Dahlem (Figure 7.17) | 0 | 30 | 4414 | 390 | Figure 7.22 |
| Freiburg (Figure 7.18) | 0 | 30 | 6555 | 1476 | Figure 7.23 |

Table 7.1: General information of the road network experiments. The demonstrated travel distance and duration are an average of those covered by the motion planner and the path planner. The slight variation of the coverages of the two planners can be ignored.

$$\delta s_{box} = (t_{red-start-assumed} - t_{scan-end}) \times v_{assumed} \tag{7.5}$$

where $v_{assumed}$ refers to the speeds of the obstacles from the perspective of the planner. In comparison, the displacements $\delta s_{obstacle}$ of the displayed real obstacles from where they were at $t_{scan-begin}$ are:

$$\delta s_{obstacle} = (t_{red-start-assumed} + \delta t_{passed-from-red-start-assumed} - t_{scan-begin}) \times v_{real} \tag{7.6}$$

where $v_{real}$ refers to the real speeds of the obstacles. It is straightforward that $\delta t_{scan-generate}$, $\delta t_{passed-from-red-start-assumed}$ and the difference between $v_{assumed}$ and $v_{real}$ result in the location differences between the displayed cyan boxes and real obstacles. In future work, $\delta t_{scan-generate}$ should be taken into account in the specification of the time when the obstacles are scanned.

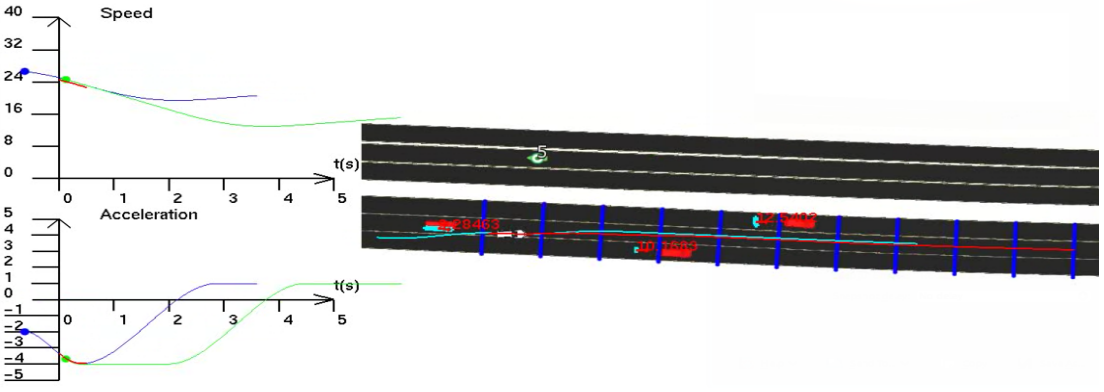



Figure 7.20: The vehicle traverses through the moving traffic along road network AVUS. See the notes of Figure 7.10 and Figure 7.11 for the meanings of the points and curves.

| Track | Distance jerk level (motion planner, expected-jerk based) | Distance jerk level (motion planner, simulation-based) | Distance jerk level (path planner, simulation-based) | Heading jerk level (motion planner) | Heading jerk level (path planner) |
|---|---|---|---|---|---|
| Avus-0 | 0.46 | 63 | 61 | 0.173 | 0.000833 |
| Avus-1 | 0.34 | 65 | 21 | 0.00235 | 0.000105 |
| Avus-2 | 0.54 | 59 | 105 | 0.0049 | 0.000124 |
| Dahlem | 0.59 | 35 | 15 | 0.0008 | 0.000159 |
| Freiburg | 0.358 | 136 | 30 | 0.0033 | 0.00027 |

Table 7.2: Jerk levels of the trajectories generated by the proposed motion planner and the path planner in driving the vehicle in road networks. The demonstrated jerk level is generated by normalizing the jerk level calculated based on Equation 7.4 by the travel duration.

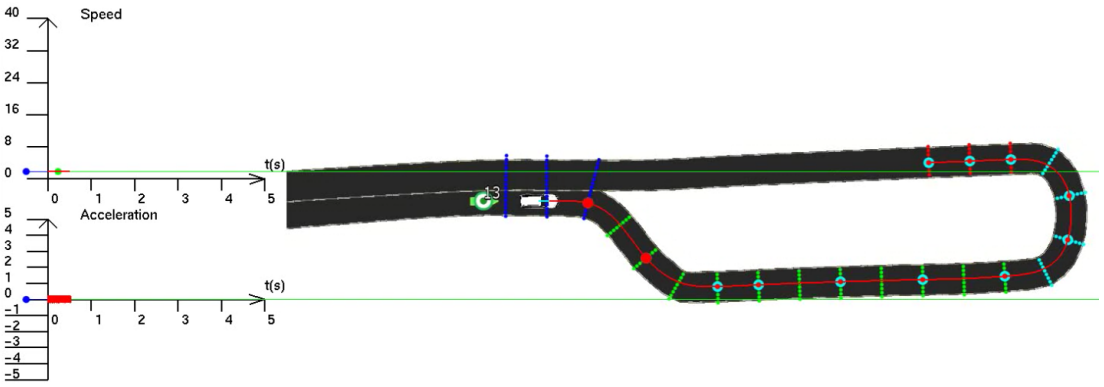



Figure 7.21: A trajectory example generated when the vehicle runs in road network Avus with no obstacles. The speed limit is $30\,m/s$. The points located on the trajectories are the spatial samples that compose the underlying paths of the trajectories. See the notes of Figure 7.10 and Figure 7.11 for the meanings of other points and curves.

## 7.4 Comparison to the State of the Art

As it is difficult to compare the performances of different motion planning strategies in the same experimental setting, the planners are compared with each other here only in terms of some specific features which play an important role in determining the performance of motion planners in handling traffic scenarios. In [23], there is a thorough discussion about the important features of motion planners and a detailed comparison of the motion planners that exist at the moment when the work proposed in [23] came into being. Figure 7.24 compares different motion planners that are still active in the literature and relatively mature at the moment when this thesis is written.

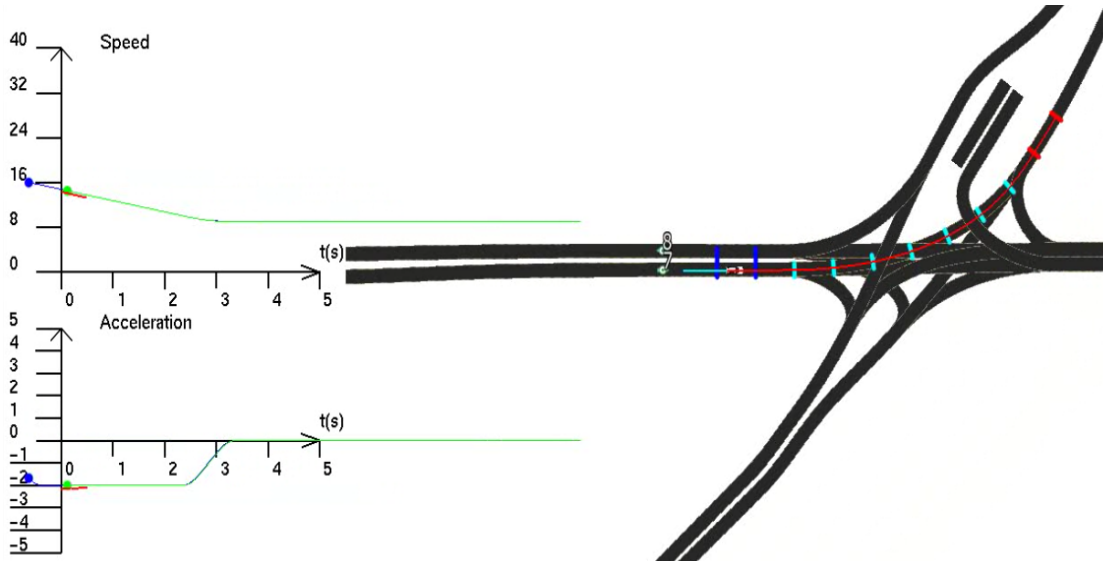



Figure 7.22: A trajectory example generated when the vehicle runs in road network Dahlem with no obstacles. The speed limit is 30 $m/s$. See the notes of Figure 7.21 for the meanings of the points and curves.

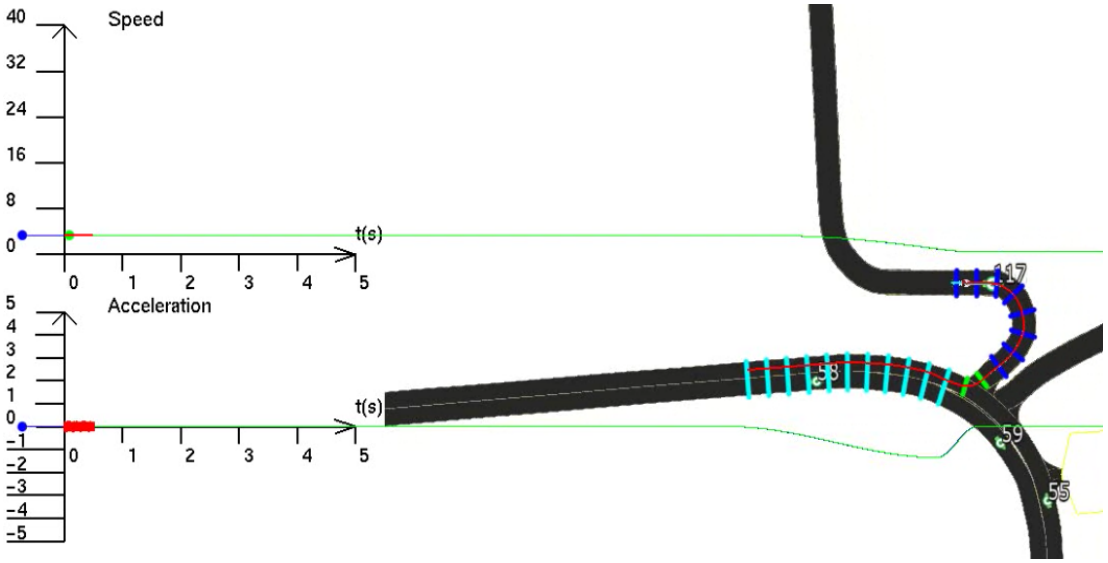



Figure 7.23: A trajectory example generated when the vehicle runs in road network Freiburg with no obstacles. The speed limit is 30 $m/s$. See the notes of Figure 7.21 for the meanings of the points and curves.

| | Proposed planning strategy | CMU/ Mcnaughton | CMU/ Xu | CMU/ Gu | Karlsruhe & Stanford |
|---|---|---|---|---|---|
| Deliberative approach; explicitly evaluate candidate trajectories | Yes | Yes | Yes | Yes | Yes |
| Contain multiple lateral shifts in one trajectory | Yes | Yes | Yes | Yes | Yes |
| Contain multiple phases of deceleration and acceleration in one trajectory | Yes | Yes | Yes | Yes | Yes |
| Possibility of scaling performance with parallel computation | Yes | Yes | Yes | Yes | Yes |
| Hard real-time response | Yes | Yes | Yes | Yes | Yes |
| Planing consistently in road networks with various road layouts | Yes | Some | N/A | N/A | Yes |
| Smooth trajectories | Yes | Some | Yes | Yes | Yes |

Figure 7.24: Comparison of the proposed planner (the fist column) against the state of the art. "CMU/Mcnaughton" , "CMU/Xu", "CMU/Gu" refer to the works proposed in [23], [25] and [26] respectively. "Karlsruhe & Stanford" refers to the similar planners presented in [17] and [15].

## 7.5 Summary

This chapter assessed the proposed motion planner according to the common criteria for evaluating motion planning strategies. Some experiments were carried out to examine the capability of the proposed planner in dealing with time-critical traffic scenarios and navigating the vehicle around the road networks. A comparative evaluation was also conducted to compare the proposed planner with the state of the art.

As is observed in the experiments, the generation of the simulated sensor data is slowed down by the application of the motion planner. That is because both of the sensor simulation and the state lattice construction exploit the computational resources of the GPU. Future work may consider using separate GPU devices for these two tasks.

It should be pointed out that the realistic level of the simulation determines the reliability of the validation result. As mentioned earlier, some physical properties of the vehicle that are demonstrated when it is running at high speed are not yet taken into account in the simulation model of the vehicle. Consequently, the cost functions and their parameters adopted in the current implementation of the planner might still be subject to further adaptations according to the result of real-life tests. Besides, the absence of a noise model in the simulated sensors contributes to the unrealistic level of the sensor data, which should be addressed in future work.

In sum, the proposed motion planner is capable of generating flexible and feasible trajectories in most experiments. However, the computational complexity is still not so satisfactory which requires further improvements.

# Chapter 8

# Conclusions

## 8.1 Conclusions

The proposed motion planner can generate reasonable plans in most simulated traffic scenarios. It employs a state lattice to construct tens of thousands of candidate trajectories and selects the best constraint-abiding one based on a set of cost criteria. The candidate trajectories can vary from one represented by a straight path associated with a single acceleration profile to one containing multiple lateral shifts and multiple phases of acceleration and deceleration. Such trajectory diversity can help to enhance the flexibility of the planner and the extent of global optimality of its plans.

The spatial planning horizon specified by the planner can have non-uniform width along its lateral reference. That is, it can consist of multiple segments whose widths can be different from each other. This feature helps the planner to adapt to various road layouts easily and to generate plans consistently.

During the construction of the state lattice, the path edges are associated with the acceleration profiles to generate trajectory edges. The types of the applied acceleration profiles and the association strategy play an important role in determining the feasibility of the constructed trajectories. Acceleration cubic polynomials and constant accelerations are applied in the proposed planner. The adopted association scheme makes it possible to span one acceleration profile over several trajectory edges, which, together with the applied smooth acceleration profiles, helps to enhance the feasibility of the trajectories.

In the construction of cost maps of obstacles for the trajectory evaluation, the obstacles need to be dilated to compensate for the vehicle shape. The dilation scheme proposed in [23] is adopted in this work. The main idea of this dilation strategy is that the maximum deviation of the heading direction of the vehicle from parallel to the center line of the travelling road is assumed to be six degrees. A novel approach is proposed in this work to analyse the sufficiency of this dilation strategy from the perspective of excluding all the trajectories that are practically not traversable.

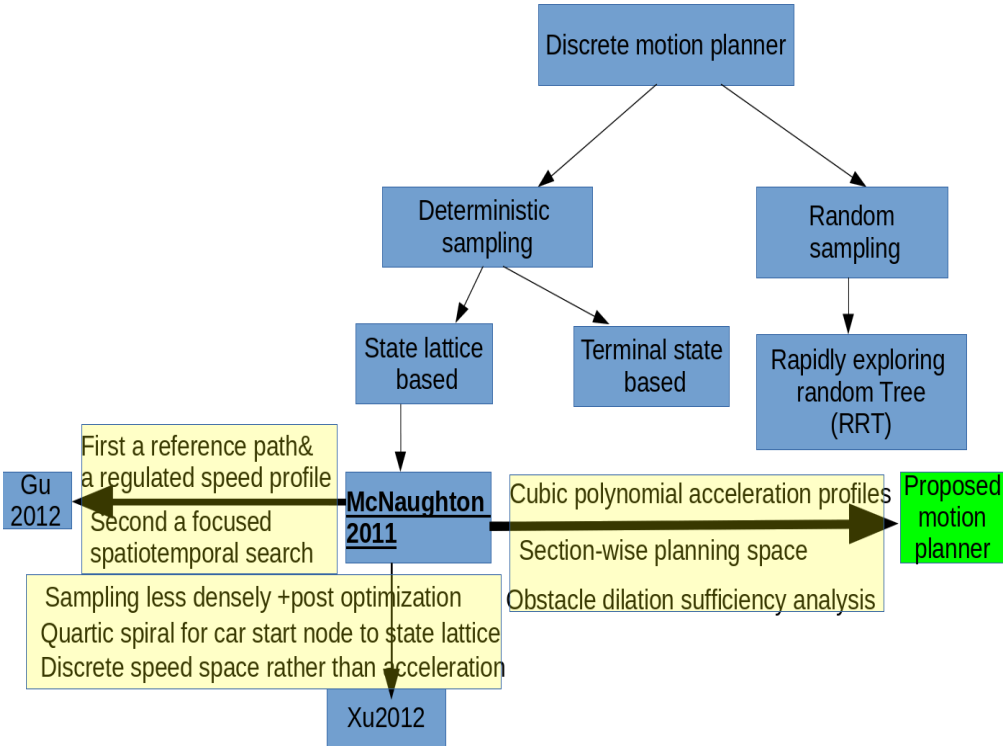



Figure 8.1: A general overview of variants of planners under the category of discrete planners."Mcnaughton2011" , "Xu2012", "Gu2012" refer to the works proposed in [23], [25] and [26] respectively.

Figure 8.1 shows the a general overview of variants of planners(including the proposed planner) under the category of discrete planners.

As the experiments in which the performance of the proposed planner is evaluated are conducted in a simulation environment, the reliability of the evaluation depends largely on the realistic level of the simulation. To that end, the scanning sensors are also simulated for the experiments. The simulated sensor data can be used by the obstacle detection and tracking modules in the same way as the real-life sensor data. In this way, more realistic judgements about the surrounding traffic can be generated which can help to explore the real performance of the planner. The simulation uses the programmable shaders in the rendering pipeline of OpenGL to record sensor-related data. Such implementation takes advantage of the parallel processing power of the GPU and thus enhances the computational efficiency of the generation process of the simulated sensor data. A novel macro-micro approach is proposed which can increase the accuracy of the simulated sensor data without hampering the computational efficiency.

An obvious next step is to design and embed a scenario reasoning module in the current planning architecture. Such scenario reasoning module is necessary for providing far-sighted guidance for the short-sighted planner. In addition, the current obstacle prediction module should be further improved in order to provide sufficient prediction

of the future route of the traffic, which is crucial for the effectiveness and safety of the trajectories generated by the motion planner. Besides, a more sophisticated controller that can translate a constraint-abiding plan into a sequence of effective actuation commands is also necessary. As is mentioned in Chapter 5, the controller designed in this work can only be effective in controlling a nonholonomic vehicle model. Consequently, it cannot be tested whether the trajectories generated by the proposed planner can be well followed by the vehicle dynamics at very high speed. Theoretically, as long as the plans satisfy the constraints of vehicle dynamics, they should be correctly executed by the vehicle. What will exactly happen practically and what constraints should be taken into account are still subject to further investigations. Only with a high advanced vehicle model or even a real vehicle can the limits of the vehicle and the planner be further tested. In order for more realistic experiments to be carried out safely and effectively, a sound scenario reasoning module, a good enough module for predicting the future routes of the traffic participants and an advanced controller are necessary. The feedbacks gathered in the experiments can guide the further improvements of the proposed planner.

## 8.2 Future Work

Several advices have been suggested for future work in the previous chapters. The main ideas of them are summarised in the following.

### 8.2.1 Adaptive State Lattice

The state lattice is constructed based on the spatial horizon. The principles that guide the current method of specifying the spatial horizon might sometimes result in a spatial horizon with segments that are unnecessarily short. Specific heuristics can be designed to remove such segments in future work.

A non-uniform sampling of the spatial horizon should be considered for the construction of the state lattice in future work. Such non-uniform sampling should make sure that all the center lines of the lanes in the spatial horizon should be sampled, which is beneficial in improving the consistency of successive state lattices. It should also guarantee that it is easy to incorporate arbitrary position into the set of spatial nodes, which is useful for designing trajectories that can reach a target speed at a target location.

It is necessary to make the sampling units and connectivity patterns of the state lattice adjustable in runtime. In this way, a feasible and traversable trajectory can be discovered as much as possible, which helps to guarantee a certain level of completeness.

### 8.2.2 Efficient, Effective and Consistent Cost Maps

The analysis of the sufficiency of the assumed amount of dilation needs to be automated. Once it is automated, it can be employed to discover the minimum number of dilations that are required to provide sufficient safety checking. Based on such information,

a minimum number of cost maps are necessary to be constructed. In this way, the efficiency and effectiveness of the cost maps can be improved.

Further, it is better to apply adaptive sizes of the submaps of the cost maps. The small, fixed submap size applied in the current implementation of the proposed planner is not necessary when, for example, the vehicle drives on a straight road.

Measures should be taken to guarantee a certain level of consistency between the cost maps for successive planning horizons where it is possible to do so. This is helpful in improving planning consistency.

### 8.2.3 Trajectory Feasibility

Only curvature cubic polynomials are applied in the current implementation of the proposed planner. Higher-degree polynomials should be considered for representing the path edges as the trajectories composed of them can achieve the continuity in higher-order derivatives of the curvature with respect to arclength.

As for the speed trajectory, more diversified acceleration profiles should be considered, such as the acceleration profile for the vehicle following behaviour. The practical duration necessary for the vehicle to transit from one acceleration to another needs to be further identified.

A lightweight post-optimization of the generated trajectories should be implemented to further increase their smoothness.

A proper compensation for the planning latency is necessary for guaranteeing a certain level of consistency between the trajectories generated in successive planning horizons. At the moment, the amount of compensation is fixed, which is a flawed assumption. Heuristics need to be discovered that can be used to predict the planning duration using the information such as the starting state of the vehicle and the road layout. More accurate compensations can thus be applied. Besides, the latency caused by the controller and the vehicle's execution of the actuation commands should be identified and considered in the forward simulation of the vehicle dynamics.

### 8.2.4 Computational Efficiency

In future work, all the planning phases within one planning cycle should be implemented on the GPU, such as the constructions of cost maps and path edges. The computational cost of the construction of the state lattice should be further decreased. The execution parallelism and the instruction output of the threads launched for constructing and evaluating the trajectory edges need to be further maximized in order to make full use of the parallel computing power of the GPU.

Regarding the simulation experiment, the frequency of the generation of the simulated sensor data is still intolerably low when the number of objects that are present in the simulated scenario is large. Simulation strategies that can further decrease the computational cost should be developed.

As both of the planner and the simulation of the scanning sensors exploit the computational resources provided by the GPU, when they work at the same time, both of them are somehow slowed down. Consequently, future work can consider executing the two tasks on separate GPUs.

### 8.2.5 Realistic Simulation

The simulation consists of modelling the vehicle dynamics, the traffic environment and the scanning sensors. These three aspects all require further improvements.

The simulation of the vehicle dynamics should take into account the complicated vehicle dynamics at high speed, the delay in executing the actuation commands, etc.

Regarding the simulation of the traffic environments, more realistic interactions between the traffic participants should be considered.

In terms of the simulation of the scanning sensors, the first task might be validating the simulated sensor data against the real-life sensor data. Should the discrepancy between the simulated and virtual sensor data turn out to be intolerably large, a full-featured simulation that can consider the potential effect of physical phenomena on the returned signals can be designed. Another alternative is to simulate the sensor data based on given ground truth data. Furthermore, noises should be added to the simulated sensor data to increase the realistic level of uncertainty of the sensing system.

### 8.2.6 Realistic and Complicated Experiments

As is mentioned in the previous section, a scenario reasoning module which can provide long-term goals for the planner to follow is necessary for adapting the behaviour of the planner to scenario-dependant requirements. Embedding the scenario reasoning module in the planning architecture is a task in future work. The embedding strategy that is suggested in Section 5.3 can be employed. To design an effective scenario reasoning module, the requirements of various traffic scenarios should be investigated. Besides, testing and adjusting the optimal parameters of the cost functions for each scenario requires a lot of engineering efforts. To that end, this process needs to be automated using techniques such as machine learning.

Besides, a module that can give a rather accurate prediction of the future route of the moving traffic is necessary. Such module is essential for an effective motion planner. As a matter of fact, one of the major advantages of a motion planner in comparison with a path planner is its capability in handing time-critical traffic scenarios, where an accurate prediction of the future trajectories of the traffic participants is indispensible.

In addition, a more advanced controller is necessary that can consider the complicated vehicle dynamics at high speed in its generation of the actuation commands. Further, an emergency handling module should be designed and equipped as a safety layer in the planning architecture. With a wise scenario reasoning module, a sophisticated

controller and a powerful emergency handling module, more realistic and complicated experiments can be effectively and safely carried out.

# Bibliography


[1] B Fischer, T Ganjineh, D Göhring, S Hempel, T Langner, D Latotzky, A Reuschenbach, M Schnürmacher, E Tapia, F Ulbrich, et al. Technical report autonomos 2010. Technical report, tech. rep., Freie Universität Berlin, 2011.

[2] Bart Dewulf, Tijs Neutens, Mario Vanlommel, Steven Logghe, Philippe De Maeyer, Yves De Weerdt, and Nico Van de Weghe. Examining daily commuting patterns using gis. In *NECTAR Cluster 6, Meeting abstracts*. Network on European Communications and Transport Activities Research (NECTAR), 2014.

[3] Alois Stutzer and Bruno S Frey. Stress that doesn't pay: The commuting paradox*. *The Scandinavian Journal of Economics*, 110(2):339–366, 2008.

[4] David Schrank, Bill Eisele, and Tim Lomax. Tti's 2012 urban mobility report. *Texas A&M Transportation Institute. The Texas A&M University System*, 2012.

[5] Wolfgang Knospe, Ludger Santen, Andreas Schadschneider, and Michael Schreckenberg. Human behavior as origin of traffic phases. *Physical Review E*, 65(1):015101, 2001.

[6] Huei Peng. Ground vehicle active safety systems. In *The Fifth annual Conference of the Automotive Research Center*.

[7] Thierry Fraichard and Hajime Asama. Inevitable collision states—a step towards safer robots? *Advanced Robotics*, 18(10):1001–1024, 2004.

[8] Statistisches Bundesamt. Final report: Energy consumption and co2 emissions of road transport as part of the namea compilation strategy. *Statistisches Bundesamt, Wiesbaden*, 2012.

[9] David J Thurman, Ettore Beghi, Charles E Begley, Anne T Berg, Jeffrey R Buchhalter, Ding Ding, Dale C Hesdorffer, W Allen Hauser, Lewis Kazis, Rosemarie Kobau, et al. Standards for epidemiologic studies and surveillance of epilepsy. *Epilepsia*, 52(s7):2–26, 2011.

[10] Sören Kammel, Julius Ziegler, Benjamin Pitzer, Moritz Werling, Tobias Gindele, Daniel Jagzent, Joachim Schröder, Michael Thuy, Matthias Goebl, Felix von Hundelshausen, et al. Team annieway's autonomous system for the 2007 darpa urban challenge. *Journal of Field Robotics*, 25(9):615–639, 2008.

[11] Jonathan Bohren, Tully Foote, Jim Keller, Alex Kushleyev, Daniel Lee, Alex Stewart, Paul Vernaza, Jason Derenick, John Spletzer, and Brian Satterfield. Little ben: The ben franklin racing team's entry in the 2007 darpa urban challenge. *Journal of Field Robotics*, 25(9):598–614, 2008.

[12] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 25(9):569–597, 2008.

[13] J Rojo, R Rojas, K Gunnarsson, M Simon, F Wiesel, F Ruff, L Wolter, F Zilly, N Santrac, T Ganjineh, et al. Spirit of berlin: An autonomous car for the darpa urban challenge-hardware and software architecture. *Technical semifinalist paper of DARPA Urban Challenge*, 2007.

[14] Chris Urmson, Chris Baker, John Dolan, Paul Rybski, Bryan Salesky, William Whittaker, Dave Ferguson, and Michael Darms. Autonomous driving in traffic: Boss and the urban challenge. *AI Magazine*, 30(2):17, 2009.

[15] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 163–168. IEEE, 2011.

[16] R Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, DTIC Document, 1992.

[17] Moritz Werling, Sören Kammel, Julius Ziegler, and Lutz Gröll. Optimal trajectories for time-critical street scenarios using discretized terminal manifolds. *The International Journal of Robotics Research*, 31(3):346–359, 2012.

[18] Mihail N Pivtoraiko. Differentially constrained motion planning with state lattice motion primitives. Technical report, DTIC Document, 2012.

[19] Ken Goldberg. Completeness in robot motion planning. In *The First Workshop on Algorithmic Foundations of Robotics*, pages 419–430. Citeseer, 1994.

[20] Peng Cheng and Steven M LaValle. Resolution completeness for sampling-based motion planning with differential constraints. *International Journal of Robotics Research*, 2004.

[21] Petr Svestka. *On probabilistic completeness and expected complexity of probabilistic path planning*. Citeseer, 1996.

[22] Luis Martinez-Gomez and Thierry Fraichard. Collision avoidance in dynamic environments: an ics-based solution and its comparative evaluation. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 100–105. IEEE, 2009.

[23] Matthew McNaughton. Parallel algorithms for real-time motion planning. 2011.

[24] Julius Ziegler and Christoph Stiller. Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1879–1884. IEEE, 2009.

[25] Wenda Xu, Junqing Wei, John M Dolan, Huijing Zhao, and Hongbin Zha. A real-time motion planner with trajectory optimization for autonomous vehicles. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2061–2067. IEEE, 2012.

[26] Tianyu Gu and John M Dolan. On-road motion planning for autonomous vehicles. In *Intelligent Robotics and Applications*, pages 588–597. Springer, 2012.

[27] Tianyu Gu, Jarrod Snider, John M Dolan, and Jin-woo Lee. Focused trajectory planning for autonomous on-road driving. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 547–552. IEEE, 2013.

[28] Neville Hogan. An organizing principle for a class of voluntary movements. *The Journal of Neuroscience*, 4(11):2745–2754, 1984.

[29] Matt Visser. Jerk, snap and the cosmological equation of state. *Classical and Quantum Gravity*, 21(11):2603, 2004.

[30] Toshiaki Kono, Takumi Fushiki, Katsuya Asada, and Kenjiro Nakano. Fuel consumption analysis and prediction model for eco route search. In *15th World Congress on Intelligent Transport Systems and ITS America's 2008 Annual Meeting*, 2008.

[31] Matthew Barth, Sindhura Mandava, Kanok Boriboonsomsin, and Haitao Xia. Dynamic eco-driving for arterial corridors. In *Integrated and Sustainable Transportation System (FISTS), 2011 IEEE Forum on*, pages 182–188. IEEE, 2011.

[32] Ben Ezair, Tamir Tassa, and Zvi Shiller. Planning high order trajectories with general initial and final conditions and asymmetric bounds. *The International Journal of Robotics Research*, 33(6):898–916, 2014.

[33] Shuiying Wang and Raúl Rojas. Shader-based automatic camera layout optimization for mobile robots using genetic algorithm. In *GRAPP*, pages 153–160, 2014.

[34] Jean-Claude Latombe. Robot motion planning, chapter. 1996.

[35] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[36] Michael Brady. *Robot motion: Planning and control*. MIT press, 1982.

[37] Dave Ferguson, Thomas M Howard, and Maxim Likhachev. Motion planning in urban environments. *Journal of Field Robotics*, 25(11-12):939–960, 2008.

[38] Matthew McNaughton, Chris Urmson, John M Dolan, and Jin-Woo Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4889–4895. IEEE, 2011.

[39] Daniel Althoff, Martin Buss, Andreas Lawitzky, Moritz Werling, and Dirk Wollherr. On-line trajectory generation for safe and optimal vehicle motion planning. In *Autonomous Mobile Systems 2012*, pages 99–107. Springer, 2012.

[40] Steven M LaValle. Rapidly-exploring random trees a new tool for path planning. 1998.

[41] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1): 116–129, 2002.

[42] Yoshiaki Kuwata, Gaston A Fiore, Justin Teo, Emilio Frazzoli, and Jonathan P How. Motion planning for urban driving using rrt. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1681–1686. IEEE, 2008.

[43] John Leonard, Jonathan How, Seth Teller, Mitch Berger, Stefan Campbell, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Albert Huang, Sertac Karaman, et al. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10): 727–774, 2008.

[44] Peng Cheng and Steven M LaValle. Reducing metric sensitivity in randomized trajectory design. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 43–48. IEEE, 2001.

[45] Anna Atramentov and Steven M LaValle. Efficient nearest neighbor searching for motion planning. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, pages 632–637. IEEE, 2002.

[46] Alonzo Kelly and Bryan Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *The International Journal of Robotics Research*, 22 (7-8):583–601, 2003.

[47] A Segovia, M Rombaut, A Preciado, and D Meizel. Comparative study of the different methods of path generation for a mobile robot in a free environment. In *Advanced Robotics, 1991.'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on*, pages 1667–1670. IEEE, 1991.

[48] M Tounsi and JF Le Corre. Trajectory generation for mobile robots. *Mathematics and computers in simulation*, 41(3):367–376, 1996.

[49] Rodney A Brooks. Solving the find-path problem by good representation of free space. *Systems, Man and Cybernetics, IEEE Transactions on*, (2):190–197, 1983.

[50] Kiyoshi Komoriya, Susumu Tachi, and Kazuo Tanie. A method of autonomous locomotion for mobile robots. *Advanced Robotics*, 1(1):3–19, 1986.

[51] Hiroshi Akima. Algorithm 433: interpolation and smooth curve fitting based on local procedures [e2]. *Communications of the ACM*, 15(10):914–918, 1972.

[52] A Takahashi, Takero Hongo, Yoshilci Ninomiya, and Gunji Sugimoto. Local path planning and motion control for agv in positioning. In *Intelligent Robots and Systems' 89. The Autonomous Mobile Robots and Its Applications. IROS'89. Proceedings., IEEE/RSJ International Workshop on*, pages 392–397. IEEE, 1989.

[53] Fabian Schwarzer, Mitul Saha, and Jean-Claude Latombe. Exact collision checking of robot paths. *Algorithmic foundations of robotics V*, pages 25–42, 2004.

[54] P Jiménez, F Thomas, and C Torras. Collision detection algorithms for motion planning. In *Robot Motion Planning and Control*, pages 305–343. Springer, 1998.

[55] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28 (8):933–945, 2009.

[56] Julius Ziegler and Christoph Stiller. Fast collision checking for intelligent vehicle motion planning. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 518–522. IEEE, 2010.

[57] Dave Ferguson and Maxim Likhachev. Efficiently using cost maps for planning complex maneuvers. *Lab Papers (GRASP)*, page 20, 2008.

[58] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317. IEEE, 1994.

[59] Anthony Stentz. The focussed dˆ* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995.

[60] Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *ICAPS*, pages 262–271, 2005.

[61] P Haslum H Geffner and P Haslum. Admissible heuristics for optimal planning. In *Proceedings of the 5th Internat. Conf. of AI Planning Systems (AIPS 2000)*, pages 140–149, 2000.

[62] Hans-Ullrich Doehler and Dirk Bollmeyer. Simulation of imaging radar for obstacle avoidance and enhanced vision. In *AeroSense'97*, pages 64–72. International Society for Optics and Photonics, 1997.

[63] C. Berger and B. Rumpe. Nutzung von projektiven texturen auf einer gpu zur distanzmessung für automotive sensorsimulationen. In *Proceedings des*, volume 10, 2009.

[64] Andrew S Glassner. *An introduction to ray tracing*. Morgan Kaufmann, 1989.

[65] Timothy J Purcell, Ian Buck, William R Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics (TOG)*, 21(3): 703–712, 2002.

[66] Sven Woop, Jörg Schmittler, and Philipp Slusallek. Rpu: a programmable ray processing unit for realtime ray tracing. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 434–444. ACM, 2005.

[67] G.L. Bair. Airborne radar simulation. *Camber Corporation, Dallas, Texas*, 1996.

[68] N. Peinecke, H.U. Doehler, and B.R. Korn. Simulation of imaging radar using graphics hardware acceleration. In *Proceedings of SPIE, the International Society for Optical Engineering*, pages 69570L–1. Society of Photo-Optical Instrumentation Engineers, 2008.

[69] N. Peinecke, T. Lueken, and B.R. Korn. Lidar simulation using graphics hardware acceleration. In *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pages 4–D. IEEE, 2008.

[70] Gazebo website http://gazebosim.org/.

[71] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004.

[72] OJ Gietelink, DJ Verburg, K Labibes, and AF Oostendorp. Pre-crash system validation with prescan and vehil. In *Intelligent Vehicles Symposium, 2004 IEEE*, pages 913–918. IEEE, 2004.

[73] http://www.nisys.de.

[74] Lars Wolter. Simulation von fahrdynamik und sensorik im stadtverkehr. Master's thesis, Freie Universität Berlin, March 2009.

[75] DARPA Urban Challenge. "route network definition file (rndf) and mission data file(mdf) formats. Technical report, tech. rep., Defense Advanced Research Projects Agency, 2007.

[76] Paul Czerwionka, Miao Wang, and Fabian Wiesel. Optimized route network graph as map reference for autonomous cars operating on german autobahn. In *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*, pages 78–83. IEEE, 2011.

[77] Franki Dillen. The classification of hypersurfaces of a euclidean space with parallel higher order fundamental form. *Mathematische Zeitschrift*, 203(1):635–643, 1990.

[78] Thomas M Howard et al. Adaptive model-predictive motion planning for navigation in complex environments. 2009.

[79] Richard Bellman and Robert E Kalaba. *Dynamic programming and modern control theory*. Academic Press New York, 1965.

[80] Paul Tompkins, Anthony Stentz, and David Wettergreen. Global path planning for mars rover exploration. In *Aerospace Conference, 2004. Proceedings. 2004 IEEE*, volume 2, pages 801–815. IEEE, 2004.

[81] Paul Tompkins. *Mission-directed path planning for planetary rover exploration*. PhD thesis, Jet Propulsion Laboratory, 2005.

[82] Tamar Flash and Neville Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *The journal of Neuroscience*, 5(7): 1688–1703, 1985.

[83] Eric W Weisstein. Cubic formula. *From MathWorld–A Wolfram Web Resource. http://mathworld. wolfram. com/CubicFormula. html*, 1999.

[84] Beng Chin Ooi, Ken J McDonell, and Ron Sacks-Davis. Spatial kd-tree: An indexing mechanism for spatial databases. In *IEEE COMPSAC*, volume 87, page 85, 1987.

[85] CUDA Nvidia. Programming guide, 2008.

[86] Team Berlin. Spirit of berlin: An autonomous car for the darpa urban challenge hardware and software architecture. *retrieved Jan*, 5:2010, 2007.

[87] Karl N Murphy. Analysis of robotic vehicle steering and controller delay. In *Fifth International Symposium on Robotics and Manufacturing (ISRAM)*, pages 631–636. Citeseer, 1994.

[88] Miao Wang. A cognitive navigation approach for autonomous vehicles. 2013.

[89] FM Lasagni. Canonical runge-kutta methods. *Zeitschrift für Angewandte Mathematik und Physik (ZAMP)*, 39(6):952–953, 1988.

[90] Shuiying Wang, Steffen Heinrich, Miao Wang, and Raúl Rojas. Shader-based sensor simulation for autonomous car testing. In *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*, pages 224–229. IEEE, 2012.

[91] Nadav Levanon. Radar principles. *New York, Wiley-Interscience, 1988, 320 p.*, 1, 1988.

[92] R. Burtch. Lidar principles and applications. In *IMAGIN Conference, Traverse City, MI*, 2002.

[93] Ibeo automotive website.

[94] HDL Velodyne. 64e lidar.

[95] R.J. Rost. *OpenGL (R) shading language*. Addison-Wesley Professional, 2005.

[96] J. Kessenich, D. Baldwin, and Randi Rost. The opengl shading language, language version 4.20, 2011.

[97] R. Wang and X. Qian. *OpenSceneGraph 3.0*. Packt Publishing, 2010.

[98] Felix Kerger. *OGRE 3D 1.7 Beginner's Guide*. Packt Publishing Ltd, 2010.

# Videos

Sensor simulation ( a combination of LiDAR sensors (Velodyne and IBEO LUX), Radar sensors and Cameras):
https://www.dropbox.com/s/wyuryn4ef60n4pc/Velodyne_Lux_radar_Camera.ogv?dl=0

Motion planner:

1) merge
https://www.dropbox.com/s/8lu9z2wtdale9lb/Merge.ogv?dl=0

2)emergency evasive
https://www.dropbox.com/s/s8cdb3c8v0zmygj/emergency_evasive.ogv?dl=0

3)Road network AVUS with obstacles
https://www.dropbox.com/s/20ys2o8hhg7uw75/3_obstacles_wholeTrack.ogv?dl=0

4)Road network Dahlem without obstacles
https://www.dropbox.com/s/libt46cgufqtim3/6_full_track_dahlem.ogv?dl=0

4)Road network Freiburg without obstacles
https://www.dropbox.com/s/4oaqq6v8ej3ywee/7_part_track_freiburg.ogv?dl=0

# Previously Published Works

Wang, Shuiying, Steffen Heinrich, Miao Wang, and Raúl Rojas. "Shader-based sensor simulation for autonomous car testing." In Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on, pp. 224-229. IEEE, 2012.

Wang, Shuiying and Raúl Rojas. "Shader-based automatic camera layout optimization for mobile robots using genetic algorithm." In GRAPP, pages 153–160, 2014.

# CV

Shuiying Wang received her bachelor's degree in mechanical engineering and automation from the University of Science and Technology Beijing in 2007 and a master's degree in aircraft flight dynamics and control from Beihang University in 2010. Supported by China Scholarship Council (2010-2014), she worked as a doctoral candidate in the artificial intelligence group of Freie Universität Berlin led by Prof. Raúl Rojas. Her research interests include simulation, motion planning, vehicle dynamics, controller design and robotics.

# Declaration of Authorship

I, Shuiying WANG, declare that this thesis titled, 'State Lattice-based Motion Planning for Autonomous On-Road Driving' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____