Development and Implementation of a Planner for Intersection Scenario for Automated Vehicle

Advanced Design Project (6 CP) No. 151/21

Editor:	Ruidi He	| 2618034
	Yanhua Zhang	| 2904036
	Yi Cui	| 2758172
	Yifei Wang	| 2906380
	Yuzhen Zhang	| 2847177
	Zhihao Liaotian	| 2897965
Supervisor:	Cheng Wang, M. Sc.	
	Kai Domhardt, M. Sc.	

TECHNISCHE UNIVERSITÄT DARMSTADT

FZD FAHRZEUGTECHNIK TU DARMSTADT

Ruidi He
Matriculation no.: 2618034
Study program: Master Mechanical and Process Engineering

Yanhua Zhang
Matriculation no.: 2904036
Study program: Master Computational Engineering

Yi Cui
Matriculation no.: 2758172
Study program: Master Computational Engineering

Yifei Wang
Matriculation no.: 2906380
Study program: Master Mechanical and Process Engineering

Yuzhen Zhang
Matriculation no.: 2847177
Study program: Master Computational Engineering

Zhihao Liaotian
Matriculation no.: 2897965
Study program: Master Mechanical and Process Engineering

Advanced Design Project (6 CP) no. 151/21
Topic:
Development and Implementation of a Planer for Intersection Scenario for Automated Vehicle

Submitted: 12th of February 2021

Technische Universität Darmstadt
Fachgebiet Fahrzeugtechnik
Prof. Dr. rer. nat. Hermann Winner
Otto-Berndt-Straße 2
64287 Darmstadt

Sworn Declaration

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB der TU Darmstadt

Hiermit versicheren wir, **Ruidi He**, **Yanhua Zhang**, **Yi Cui**, **Yifei Wang**, **Yuzhen Zhang** und **Zhihao Liaotian**, das vorliegende Advanced Design Project gemäß § 22 Abs. 7 APB TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Uns ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein gemäß § 23 Abs. 7 APD TU Darmstadt überein.

English translation for information purposes only:

Thesis Statement pursuant to § 22 paragraph 7 and § 23 paragraph 7 of APB TU Darmstadt

We herewith formally declare that we, **Ruidi He**, **Yanhua Zhang**, **Yi Cui**, **Yifei Wang**, **Yuzhen Zhang** and **Zhihao Liaotian**, have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt. we did not use any outside support except for the quoted literature and other sources mentioned in the paper. we clearly marked and separately listed all of the literature and all of the other sources, which we employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

We are aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis, the written copies and the electronic version for archiving are pursuant to § 23 paragraph 7 of APB TU Darmstadt identical in content.

Matrikelnummer: 2618034

Datum / Date: Unterschrift / Signature:

12.02.2021 Ruidi He

Matrikelnummer: 2904036

Datum / Date: Unterschrift / Signature:

12.02.2021 Yanhua Zhang

Matrikelnummer: 2758172

Datum / Date: Unterschrift / Signature:

12.02.2021 Cui

Matrikelnummer: 2906380

Datum / Date: Unterschrift / Signature:

12.02.2021 Yifei Wang

Matrikelnummer: 2847177

Datum / Date: Unterschrift / Signature:

12.02.2021 Yuzhen Zhang

Matrikelnummer: 2897965

Datum / Date: Unterschrift / Signature:

12.02.2021 Zhihao Caotian

Abstract

In this paper, a rule-based behavior planner based on a double-layer finite state machine (FSM) for the automated vehicle in an urban scenario is developed. Firstly, a universal framework for finite state machines based on the abstract factory principle is designed. To satisfy the security in different scenarios, a behavior planner is implemented on hybrid finite state machines with a precondition. The condition transitions in each finite state machine are customized with scenario features. Instead of directly commanding the desired trajectory, the behavior planner searches the safe directives for the trajectory planner, such as lateral bias and longitudinal distance. After behavior planner construction, a trajectory planner is designed, which bases on Frenet transformation, candidate waypoint random search, and trajectory cost function. A High-precision map is parsed based on the OpenDRIVE file for global planning, which represents the reference line of Frenet transformation. An optimal local trajectory is generated by a quintic polynomial, which takes full account of the safety and kinetics of the vehicle. To ensure the legality and security of generated trajectory, a decision tree is implemented at the output of the trajectory planner. Ultimately, the process of autonomous driving is illustrated in intersection scenarios using ROS/CarMaker cooperative simulation.

Table of Contents

Symbols and Indices

Latin letters:

Symbol	Unit	Term
a	$\frac{m}{s^2}$	Acceleration
A	mm^2	Surface
d, D	mm	Diameter
F	N	Force
i	./.	Ratio
a_i		Parameters of the quintic polynomial
x		Cartesian coordinate
r		Frenet coordinate
d	m	Lateral movement in Frenet coordinate
s	m	Longitudinal movement in Frenet coordinate
t		Tangential vector
n		Normal vector
v	m/s	Velocity of the vehicle
M		Time matrix of the Quintic-Polynomial
T	s	Time interval
J	m/s^3	Integral of jerk
C		Cost function
k		Weight

Greek letters:

Symbol	Unit	Term
θ	°, rad	Angle of rotation
κ	1/m	Curvature
τ	s	End time

Indices:

Symbol	Term
r	Frenet coordinate
x	Cartesian coordinate
j	jerk

t	time
d	Lateral movement
s	Longitudinal movement
lateral	Weight of lateral trajectory
longitudinal	Weight of longitudinal trajectory

List of Abbreviations

FSM	Finite State Machine
TTC	Time-To-Collision
PET	Post Encroachment Time
XML	Extensible Markup Language
PCL	Prepare Change Left
PCR	Prepare Change Right
KL	Lane Change Left
LCL	Lane Change Left
LCR	Lane Change Right
ROS	Robot Operating System
FPR	False Positive Rate
FNR	False Negative Rate
VVE	The Virtual Vehicle Environment
CIT	The CarMaker Interface Toolbox
iTTC	Inverse Time-To-Collision

List of Figures

List of Tables

1 Introduction

Due to the gaps between intrinsic human performance limitations and physic mechanical constraints of vehicle, road, and traffic-environment characteristics[1], automated vehicles have a pivotal role in road safety. In the development of automated vehicles, driving behavior decision-making has been thought of as a critical factor in field of intelligent vehicle research. A growing body of literature recognizes the importance of path planning, which is a bridge between the behavior decision module and tracking control module of automated vehicles. However, the approaches under simplified conditions quickly lose their functions in nose-to-tail traffic scenarios and cannot meet the requirements of the high-speed driving.

In this project, a rule-based autonomous driving framework should be developed to achieve autonomous driving in highway and urban scenarios. The methodological approach taken in this project is based on V-Model. A holistic approach is utilized during development, integrating a behavior planner and trajectory planner to establish an autonomous driving framework, which bases on the double-layer finite state machine.[2] The design of the behavior planner is based on the conceptual framework proposed by Baidu Apollo.[3] And for trajectory planner, the problem of how to generate traffic-adapted trajectories must be solved. Implementation of trajectory planner depends on the procedure of Werling et. al.[4], who introduces an algorithm in combination of an optimal-control strategy within the Frenet-Frame of the street. OpenDRIVE map and CarMaker software apply a virtual validation environment for the autonomous driving framework. As a result of the restrictions caused by the COVID-19 virus, this project was carried out in the home office. Hence it does not engage with real vehicle experiments.

The project's overall structure takes the form of six chapters, including introduction, methodology, background, behavior planner design, trajectory planner design, algorithm validation, and summary with outlook.

[1] Donges, E.: Driver Behavior Models (2016).

[2] Reinholtz, C. et al.: Odin: Team VictorTango's Entry in the DARPA Urban Challenge (2009).

[3] Baidu: ApolloAuto/Apollo.

[4] Moritz Werling et al.: Optimal Trajectory Generation for Dynamic Street Scenarios in a Frenét Frame (2010).

2 Methodology

2.1 V-Model

The V-Model addresses the project cycle in a technical aspect and represents the sequence of project events.[5] The following system engineering process, as the Figure 2-1 illustrates the core activities at each level of the project cycle during system verification and validation.

At the beginning of the project process, the product requirement was defined as the behavior planner and trajectory planner development for urban intersection scenarios. The second step in this project, system engineering, was to reparation the software architecture: behavior planner on a high level to obtain the vehicle motion target; trajectory planner on a low level to generate the optimal motion trajectory. The other process steps: design, implementation, integration, and validation are explained separately in Chapters 4, 5, and 6 due to their scope.

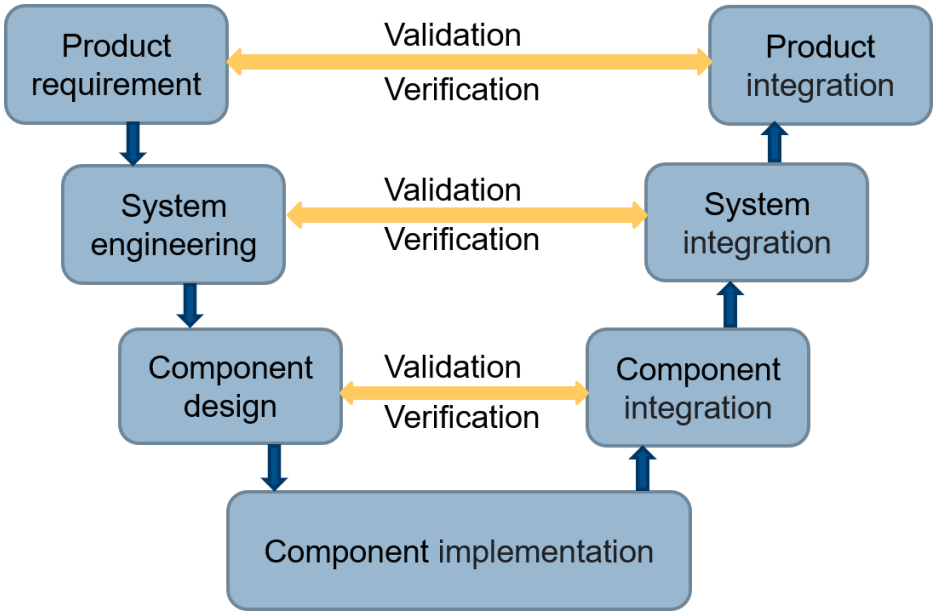


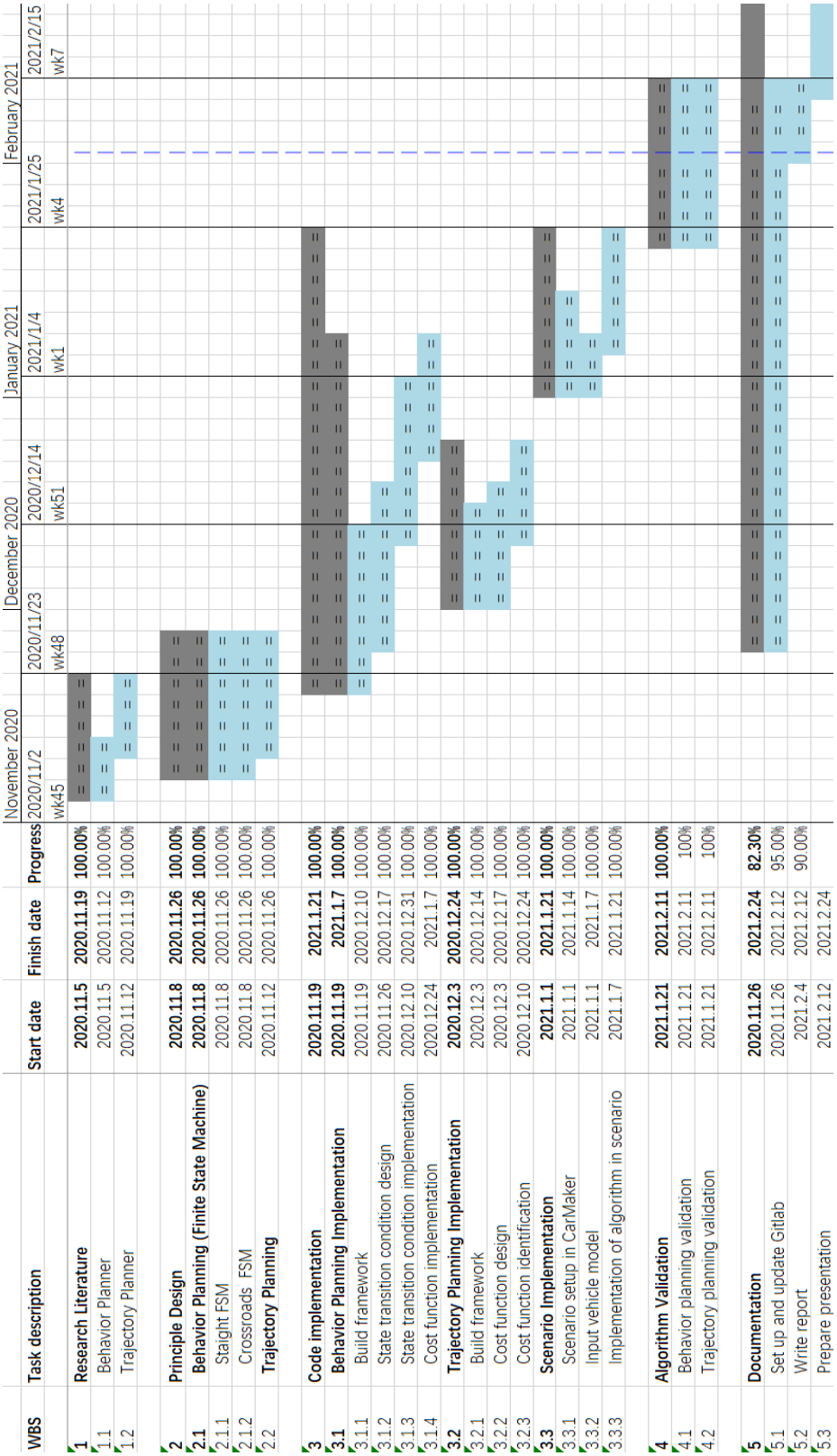
Figure 2-1: System engineering process

2.2 Project Management

Gantt chart is a bar chart that illustrates a project schedule, which also shows the dependency relationships between activities and the current schedule status.[6] The progress of the project was continuously monitored using a Gantt chart. Figure 2-2 shows an example of an excerpt from the Gantt chart, which was recorded in project week 14 with the defined subtasks for the declared acceptance of the development relationship.

[5] Forsberg, K.; Mooz, H.: The Relationship of System Engineering to the Project Cycle (1991).

[6] Clark, W. et al.: The Gantt Chart, a Working Tool of Management (1922).



WBS	Task description	Start date	Finish date	Progress
1	**Research Literature**	**2020.11.5**	**2020.11.19**	**100.00%**
1.1	Behavior Planner	2020.11.5	2020.11.12	100.00%
1.2	Trajectory Planner	2020.11.12	2020.11.19	100.00%
2	**Principle Design**	**2020.11.8**	**2020.11.26**	**100.00%**
2.1	**Behavior Planning (Finite State Machine)**	**2020.11.8**	**2020.11.26**	**100.00%**
2.1.1	Staight FSM	2020.11.8	2020.11.26	100.00%
2.1.2	Crossroads FSM	2020.11.8	2020.11.26	100.00%
2.2	**Trajectory Planning**	2020.11.12	2020.11.26	100.00%
3	**Code implementation**	**2020.11.19**	**2021.1.21**	**100.00%**
3.1	**Behavior Planning Implementation**	**2020.11.19**	**2021.1.7**	**100.00%**
3.1.1	Build framework	2020.11.19	2020.12.10	100.00%
3.1.2	State transition condition design	2020.11.26	2020.12.17	100.00%
3.1.3	State transition condition implementation	2020.12.10	2020.12.31	100.00%
3.1.4	Cost function implementation	2020.12.24	2021.1.7	100.00%
3.2	**Trajectory Planning Implementation**	**2020.12.3**	**2020.12.24**	**100.00%**
3.2.1	Build framework	2020.12.3	2020.12.14	100.00%
3.2.2	Cost function design	2020.12.3	2020.12.17	100.00%
3.2.3	Cost function identification	2020.12.10	2020.12.24	100.00%
3.3	**Scenario Implementation**	**2021.1.1**	**2021.1.21**	**100.00%**
3.3.1	Scenario setup in CarMaker	2021.1.1	2021.1.14	100.00%
3.3.2	Input vehicle model	2021.1.1	2021.1.7	100.00%
3.3.3	Implementation of algorithm in scenario	2021.1.7	2021.1.21	100.00%
4	**Algorithm Validation**	**2021.1.21**	**2021.2.11**	**100.00%**
4.1	Behavior planning validation	2021.1.21	2021.2.11	100%
4.2	Trajectory planning validation	2021.1.21	2021.2.11	100%
5	**Documentation**	**2020.11.26**	**2021.2.24**	**82.30%**
5.1	Set up and update Gitlab	2020.11.26	2021.2.12	95.00%
5.2	Write report	2021.2.4	2021.2.12	90.00%
5.3	Prepare presentation	2021.2.12	2021.2.24	

Figure 2-2: Gantt-chart in project week 14

In this project, the team member is divided into three smaller groups, taking charge of behavior planner by crossroad, behavior planner by straight road and trajectory planner.

Team A consists of Yanhua Zhang and Yuzhen Zhang, focusing on the behavior planner by the straight road. According to the V Model developing method, the requirement of this submodule is realizing behavior planner with four basic functions: Keep Lane, Change Lane, Prepare Changing Lance and Stop. The code part is based on ROS by C++ platform, implemented in relevant Publisher and Subscriber.

Team B includes Ruidi He and Yi Cui, working on the behavior planner by the crossroad. Their target aims at achieving a flexible planner, which could deal with different predefined traffic scenarios by the crossroad, for instance: stop and restart, emergency brake and “Rechts vor Links” (RvL)scenarios. Compared to the work in Team A, the reaction to the traffic signs, lights and rules were paid more attention.

Teams C is made up by Yifei Wang and Zhihao Liaotian, attending to the design and implementation of trajectory planner. The basic goal contains parsing the predefined OpenDRIVE map, generating appropriate trajectory, selection of generated trajectory based on specific cost function.

The developing of these three planners is similar and followed the V Modell method throughout the whole project. As for the Product Requirement phase, as discussed in the description above, the goal and desired functionalities were determined by all members together. Followed is the System Engineering, inside this part the framework of three planner was decided: based on the three ROS node and their communication relationships, defined by the ROS interface publisher and subscriber. Thanks to ROS architecture, once publisher and subscriber relationship was set, the content of each node were quite independent from each other. Three team could focus on own task, free from the coupling of three planner. Component Design phase includes the definition of building framework of each planner.

Component implementation is the pivotal of the program, concrete code was carried out during this phase. Our project is based on the C++ platform to fulfill the demanding performance requirements, also prepared for the future work in real-time simulation or even real test. The coding style was also defined here, such as indentation, vertical alignment, naming and comment, to keep the code more readable and modification friendly. Besides code appearance, the component of each planner was also followed the Object-oriented programming, treated as different class.

After the design, the project should pass the validation tests. Every planner should pass its own unit testing. For example, the behavior planner should be tested in the open loop, where the IPG CarMaker should take control of the auto’s motion, the behavior planner’s output command should be compared to the behavior in the CarMaker. When every planner passes the unit test, a further system testing of the whole system should be conducted, to check cooperation of these three planners.

3 Background

3.1 Framework

Autonomous vehicles have conventional vehicle functions such as acceleration, deceleration, and steering and integrate with environment perception, behavior decision-making, path planning, motion control. The core of the autonomous vehicle system's structure focuses on "intelligence", which integrates environment detection and self-care information to realize a human-like driving behavior.

The architecture of a typical autonomous vehicle system is exhibited in Figure 3-1, which is generally divided into three parts: environment perception, decision planning, and motion control.

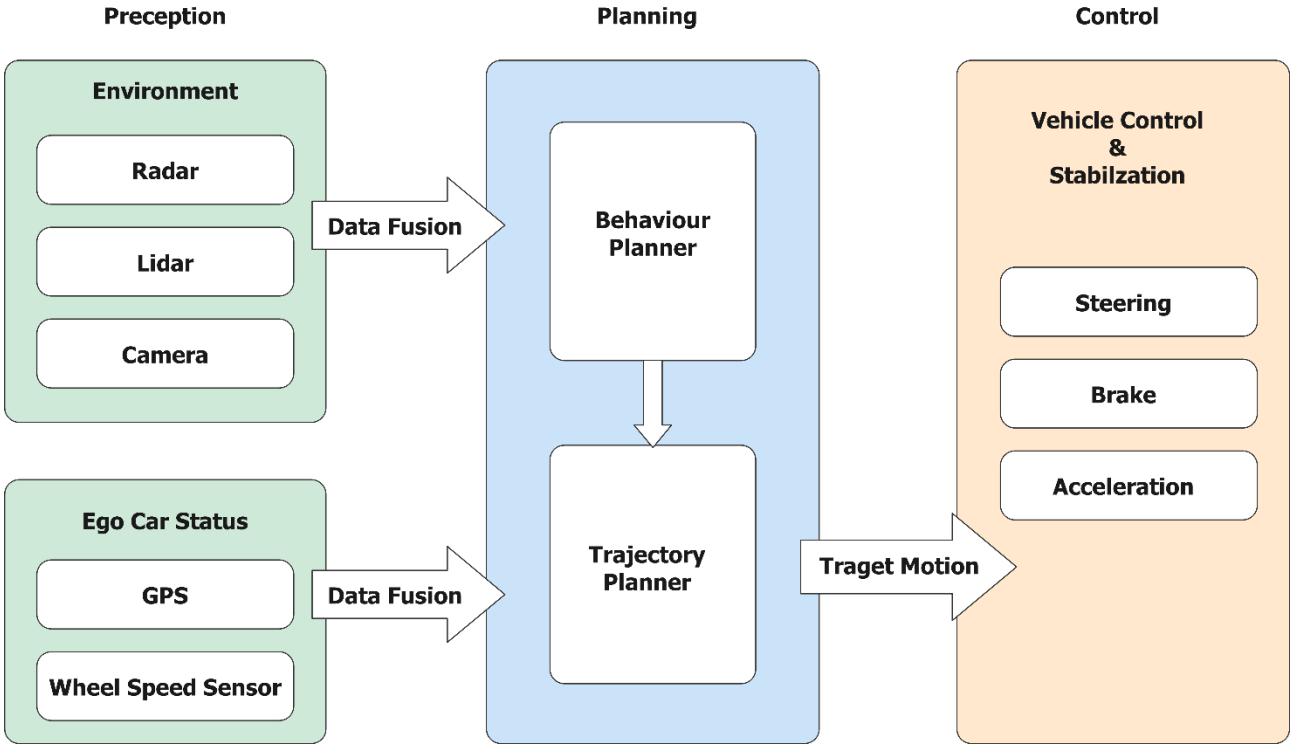


Figure 3-1: Autonomous vehicle framework

The primary purpose of the environmental perception is to obtain and process the environmental information, using multi-sensor target detection and fusion to get the information of the surrounding environment and provide these to the rest of the system. The perception layer sends the processed information to the decision planning layer. The decision planning system integrates the environment and ego car's information to produce a safe and reasonable driving behavior and guide the motion control system to control the vehicle. The behavioral decision system is a decision system in a narrow sense, which reasonably determines the current vehicle behavior based on the output information of the perception layer, determines the constraints of trajectory planning according to different behaviors, guides the trajectory planning module to plan the feasible path, speed and other information to be sent to the control layer. The motion control system receives the command from the decision planning layer and controls the vehicle response to ensure the control accuracy and track the target speed and path.

There are two main categories of behavior decision systems for the autonomous vehicle: rule-based systems and learning algorithm-based systems. Rule-based behavior planning is the method that divides the behavior of the autonomous vehicle according to the traffic rules, knowledge, experience, traffic regulations, and so on to establish a behavior rule base. Further, it divides the vehicle state using different environmental information and design rules and logic to determine vehicle behavior.

The most general rule-based behavioral decision-making method is the finite state machine method, widely applied for its clear logic and practicality, such as a series of representative applications: Junior[7], Odin[8], Talos[9]. A finite state machine is a mathematical model with a discrete input and output, also consists of a limited number of states. The current state receives events and generates corresponding actions, causing the transfer of the states. State, event, transfer, and action are the four major elements of a finite state machine. The core of a finite state machine lies in state transition. In terms of the connection logic of state transition, there are three different architectures:

- series
- parallel
- hybrid

Sub-states of the series-structured finite state machine system are connected in series, and the state transfers are mostly unidirectional and do not constitute a loop. The inputs and outputs of the sub-state in the parallel structure present a multi-node connection structure. With the different input information, the sub-state can directly access other sub-states to process and provide outputs.[10] A finite state machine system is said to have a hybrid structure if both series and parallel connections exist.

3.1.1 Series Structured Framework

The following Figure 3-2 shows the Talos framework from MIT, with an overall series structure for its behavior decision system. The decision system is constructed based on logical hierarchies, divided into modules such as localization and navigation, obstacle detection, lane line detection, road sign recognition, drivable area map construction, motion planning, and motion control, where the navigation module is responsible for decision-making tasks.

[7] Montemerlo, M. et al.: Junior: The Stanford Entry in the Urban Challenge (2008).

[8] Reinholtz, C. et al.: Odin: Team VictorTango's Entry in the DARPA Urban Challenge (2009).

[9] Leonard, J. et al.: A Perception-Driven Autonomous Urban Vehicle (2008).

[10] A. Furda; L. Vlacic: Enabling Safe Autonomous Driving in Real-World City Traffic Using Multiple Criteria Decision Making (2011).

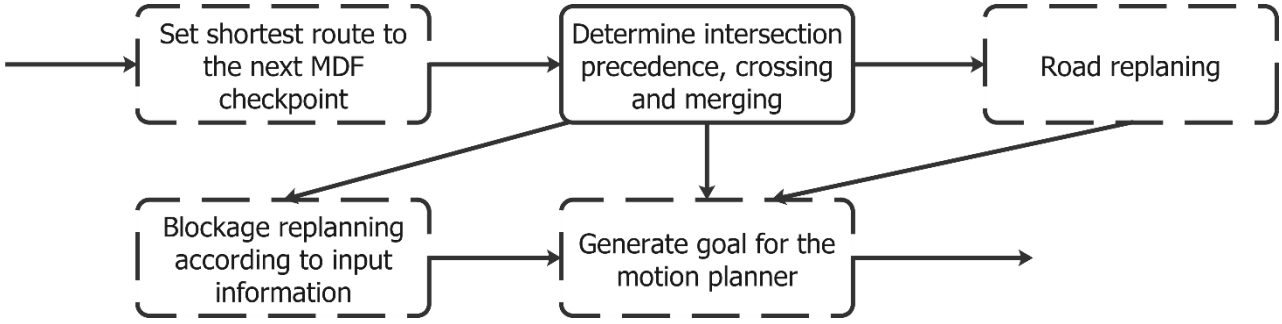


Figure 3-2: Talos: series-structured framework[11]

The advantages of the series structure are clear logic, strong planning reasoning, and high accuracy in problem-solving, but disadvantages are low adaptability to complex problems, and the failure of a sub-state can lead to the paralysis of the whole decision chain. The series structure is suitable for the specific treatment of an individual working condition and is good at hierarchical reasoning and subdivision of tasks for the solution.

3.1.2 Parallel Structured Framework

The behavior decision system of the Junior framework developed by Stanford University and Volkswagen Company is shown in Figure 3-3, which possesses a typical parallel structure. The system is divided into 13 sub-states such as locate the vehicle, forward driving, stop sign wait, intersection passing, and U-turn etc.; furthermore, each sub-state is independent of the other.

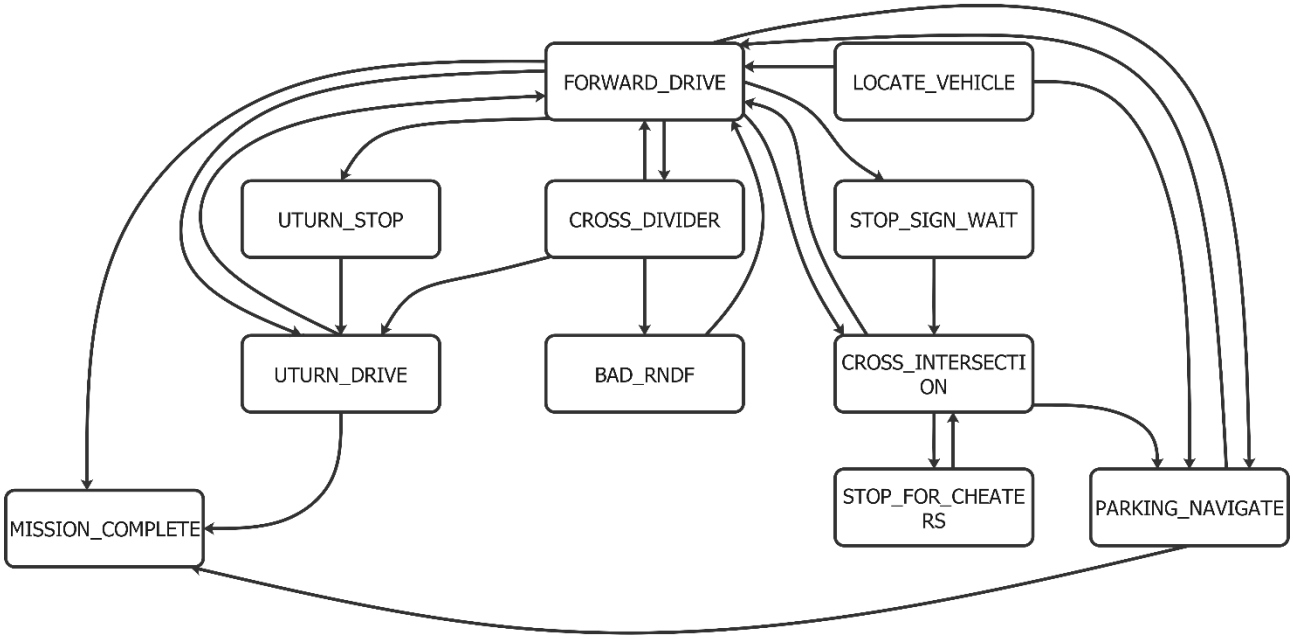


Figure 3-3: Junior: parallel-structured framework[12]

[11] Leonard, J. et al.: A Perception-Driven Autonomous Urban Vehicle (2008).

[12] Montemerlo, M. et al.: Junior: The Stanford Entry in the Urban Challenge (2008).

The decision system in Junior is one of the systems with the largest number of parallel division sub-systems. However, there are still working conditions that the finite state machine is not covered in the actual scenario test, and the accuracy of recognizing real scenarios is not ideal. It indicates that merely applying more parallel scene behavior segmentations does not guarantee an improvement of the depth of scene traversal, but on the contrary, it tends to reduce the scene recognition accuracy.

The parallel structure is suitable for more complex scenarios. Compared with the series structure, the parallel structure's advantages are the breadth of scene traversal, better modularity and expandability, and easy to realize complex function combinations. The disadvantages are that the system does not have a time sequence, lacks the depth of scene traversal, and easily ignores the subtle environmental changes in decision-making, and the grey area of state division is difficult to deal with, which leads to the wrong decision.

3.1.3 Hybrid Structured Framework

Series and parallel structures have their limits; however, the hybrid structure, a more typical method, can better combine the advantages of both.

The behavioral decision system of Odin framework developed by Virginia Tech University is shown in Figure 3-4. A decision arbitration mechanism is introduced into the system, divided into modules for lane keeping, overtaking, merging into the traffic flow, U-turn, and congestion re-planning. The outputs of each sub-decision module are submitted to the decision fuser for decision arbitration. Each module has different priorities, and the module with lower priority must yield to the module with higher priority.

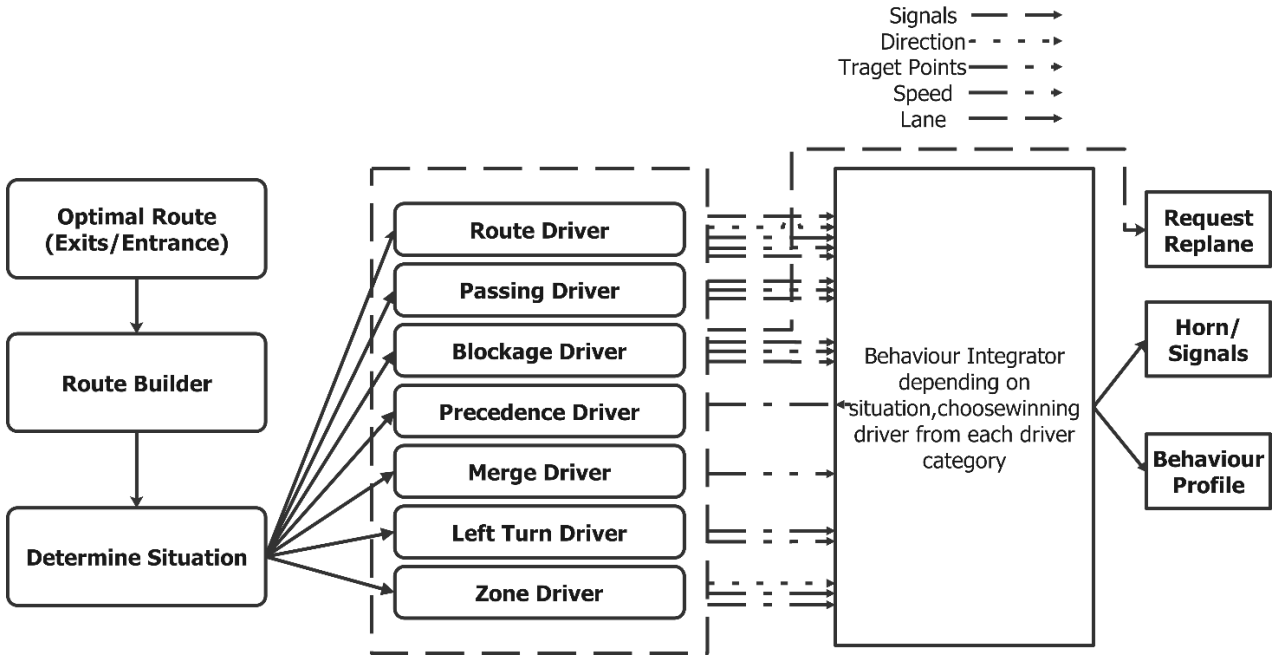


Figure 3-4: Odin: Hybrid structured framework[13]

[13] Reinholtz, C. et al.: Odin: Team VictorTango's Entry in the DARPA Urban Challenge (2009).

3.2 Surrogate Safety Indicator

In order to identify the dangerous areas, the severity of dangerous situations on a certain road needs to be evaluated, which is based on the vehicle trajectories simulation. The "surrogate safety indicators" are the parameters to evaluate the potentially dangerous interactions between vehicles and to determine which of these may lead to dangerous—or, in any case, anomalous—events. These indicators have a wide range of applications, even for the complex scenario with different road users and types of infrastructure.

Furthermore, the work of Laureshyn, Svensson and hyden[i] put forwards a new idea, which the severity of an event can be estimated by a collision risk and injury risk of an event. The factors that two kinds of risk involved are listed in table[ii], which helps the analyze of traffic more logical and reasonable.

Table 3-1: Factors affecting collision risk and injury risk

Collision risk	Injury risk
Closeness in time	Speed differences
Closeness in space	Mass differences
Speeds of the involved road users	Relative angle
	Fragility of the involved road users

Two kinds of surrogate safety indicators are applied in this Project, Time to Collision (TTC) and Post Encroachment Time (PET). TTC was applied for the first time in 1971 by Hayward[iii]. TTC is defined as the time in which two vehicles would collide if they continued travelling on the same trajectory with their speeds unchanged.

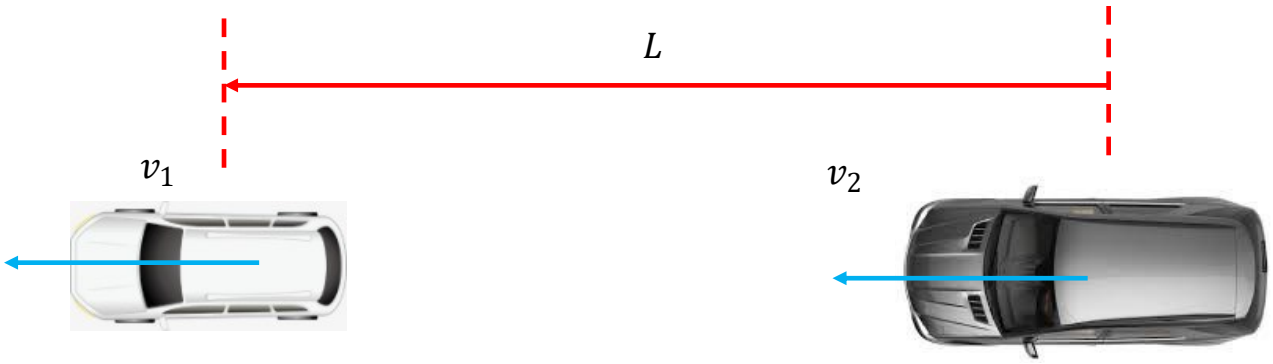


Figure 3-5: Surrogate safety indicator: TTC

TTC is an effective means of measuring traffic conflict severity and distinguishing critical behaviour from normal behavior. TTC can create the predictive trajectories for the subject vehicles and all vehicles that may interact, which makes the prediction of vehicle interactions possible.

$$TTC = \frac{L}{v_1 - v_2} \tag{3-1}$$

In 1978，Alleniv introduced the use of PET (post-encroachment time), which is the time in which the first vehicle passes at the intersection point until the moment when the second vehicle also passes under the consideration of vehicles on intersecting trajectories. PET can evaluate road safety assessment. There are also variants of PET such as gaps time, encroachment time.

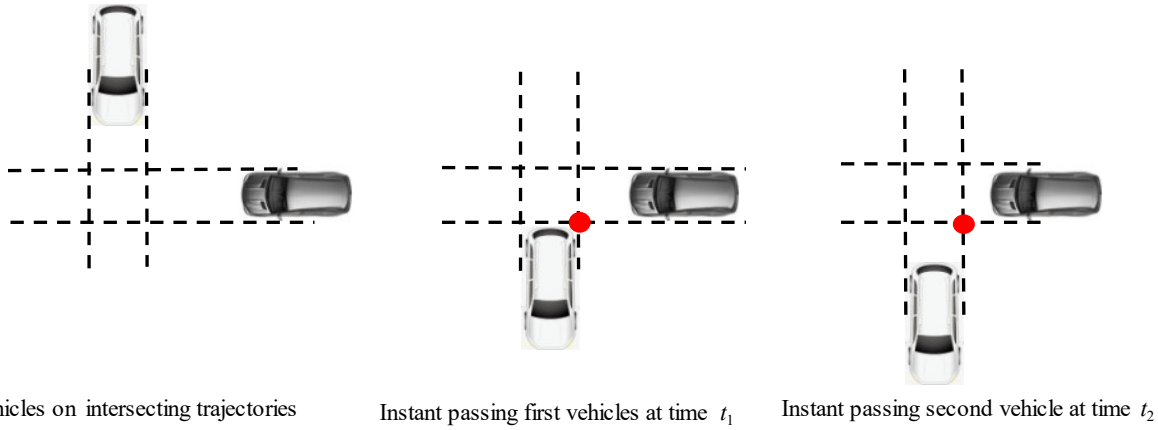


Figure 3-6: Surrogate safety indicator: PET

3.3 OpenDRIVE

OpenDRIVE is a high precision map format for road network[14], designed for vehicle navigation. OpenDRIVE focuses mainly on the simulation filed, such as automated driving simulation and revises worldwide recognition by many commercial companies, like AVL, BMW and dSPACE.Besides, many researchers base on this map format to accelerate their automated driving algorithms design simulation and certification process.

OpenDRIVE provides users with clear and detailed geometry description of required element in network: like road marking, reference line, traffic signs, road profiles, which could be also flexibly and easily modified for adaption of various road conditions. Moreover, it also offers flexible conversion to different map format, which benefits various target user-group.

These flexible functions benefit from a hierarchical structure, XML-file format. All the road segmentation elements are documents in predefined order, according to element type, attributes and so on. XML-file format plays an important role in data restoring field, due to its user-friendly and clear description. Enormous opensource XML parsers could be applied by the developer to modify, access or design their own OpenDRIVE map content easily. Many researchers and working team from auto concerns even customized their own map format, basing on the OpenDRIVE.

[14] Leitner, Andrea. et al.: Validation and Verification of Automated Systems : Results of the ENABLE-S3 Project. (2019), S. 21.

In our project, a predefined OpenDRIVE map of road test field is applied to simulate the urban road scenario. This test filed is in Griesheim and contains three junctions.

3.4 Coordinate Transformation

The traditional trajectory algorithm in the Cartesian coordinate has a favorable result on the open road, but for the urban road, ignoring the lane information leads to a high degree of path freedom and easily violates the traffic rules. Therefore, a trajectory algorithm based on the Frenet coordinate, which defines the lateral offset as the vertical distance to the centerline of the road, will be applied to the project to satisfy the scenarios and simplify the calculation while generating the trajectory.

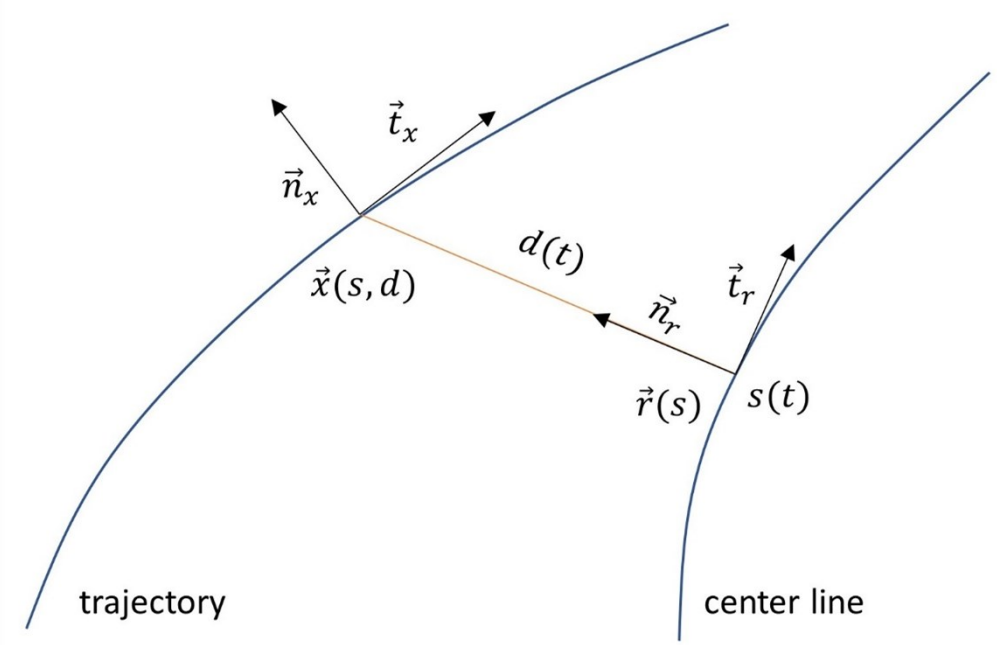


Figure 3-7: Trajectory generation in Frenet-Frame

As depicted in Figure 3-7 above, the moving reference frame is given by the tangential and normal vector $\vec{t}_r, \vec{n}_r$ at a certain point of the centerline, which is the road center in the most uncomplicated case, or the result of a path planning algorithm.[15] The original formulating of the trajectory generation in Cartesian coordinate $\vec{x}$ will be switched to the Frenet coordinate and a one-dimensional trajectory for both the root point $\vec{r}$ along the center line and the perpendicular offset d with the following relation:

$$\vec{x}\big(s(t), d(t)\big) = \vec{r}\big(s(t)\big) + d(t)\vec{n}_r\big(s(t)\big) \tag{3-2}$$

[15] M. Werling et al.: Optimal Trajectory Generation for Dynamic Street Scenarios in a Frenét Frame (2010).

As exhibited in figure 3-7, where s denotes the covered arc length of the center line and $\vec{t}_x$, $\vec{n}_x$ are the tangential and normal vectors of the trajectory $\vec{x}\big(s(t), d(t)\big)$, then a transformation mapping will be introduced in the following:

$$\left[\, \mathrm{s}, \dot{s}, \ddot{s}; d, \dot{d}, \ddot{d} \,\right] \rightarrow [\vec{x}, \theta_x, \kappa_x, v_x, a_x] \tag{3-3}$$

$$[\, \mathrm{s}, \dot{s}, \ddot{s}; d, d', d''] \rightarrow [\vec{x}, \theta_x, \kappa_x, v_x, a_x] \tag{3-4}$$

where $\dot{d}$ is derivative in time and d' is partial derivative to s.

The tangential vector $\overrightarrow{t_r}(s)$ and normal vector $\overrightarrow{n_r}(s)$ are calculated as:

$$\overrightarrow{t_r}(s) = \big[\cos\big(\theta_r(s)\big), \sin\big(\theta_r(s)\big)\big]^T \tag{3-5}$$

$$\overrightarrow{n_r}(s) = \big[-\sin\big(\theta_r(s)\big), \cos\big(\theta_r(s)\big)\big]^T \tag{3-6}$$

where $\theta_r(s)$ is the orientation of the centerline in s.

Moreover, the curvature is denoted as κ_r, so that the perpendicular offset d can be calculated as:

$$d = \big(\vec{x} - \vec{r}(s)\big)^T \overrightarrow{n_r} \tag{3-7}$$

In time domain, the first derivative of tangential vector $\dot{\overrightarrow{n_r}}$, and the first derivative of perpendicular offset $\dot{d}$ are expressed as follow:

$$\dot{\overrightarrow{n_r}} = -\kappa_r \overrightarrow{t_r} \dot{s} \tag{3-8}$$

$$\dot{d} = \big[\dot{\vec{x}} - \dot{\vec{r}}(s)\big]^T \overrightarrow{n_r} + [\vec{x} - \vec{r}(s)]^T \dot{\vec{n}}_r = v_x \sin\Delta\theta \tag{3-9}$$

By the orthogonality of vectors $[\vec{x} - \vec{r}]^T \overrightarrow{t_r} = 0$, the following formula can be derived through the derivation of time:

$$\frac{v_x}{\dot{s}} \cos\Delta\theta - 1 + \kappa_r d = 0 \tag{3-10}$$

Ultimately, the velocity in global Cartesian coordinate can be expressed as:

$$v_x = \dot{s} \frac{1 - \kappa_r d}{\cos\Delta\theta} \tag{3-11}$$

Thought the chain rule for derivatives, the partial derivative of perpendicular offset d' is inferred:

$$\begin{aligned} d' &= \frac{\mathrm{d}}{\mathrm{d}s} d = \frac{\mathrm{d}t}{\mathrm{d}s} \frac{\mathrm{d}}{\mathrm{d}t} d \\ &= \frac{1}{\dot{s}} \dot{d} \\ &= \frac{1}{\dot{s}} v_x \sin\Delta\theta \\ &= \sqrt{[1 - \kappa_r d]^2 + d'^2} \sin\Delta\theta \end{aligned} \tag{3-12}$$

After solving above equation, the partial derivative d' is expressed as:

$$d' = [1 - \kappa_r d]\tan\Delta\theta \tag{3-13}$$

Meanwhile, the second partial derivative of perpendicular offset d'' is calculated as:

$$d'' = -[\kappa_r' d + \kappa_r d']\tan\Delta\theta + \frac{1 - \kappa_r d}{\cos^2\Delta\theta}\left[\kappa_x \frac{1 - \kappa_r d}{\cos\Delta\theta} - \kappa_r\right] \tag{3-14}$$

No matter v_x is 0 or not, it can always be calculated by different formulas. Time differentiating the velocity once more yields the last unknown in transformation as follow:

$$\begin{aligned} a_x = \dot{v_x} &= \left(\dot{s}\frac{1 - \dot{\kappa}_r d}{\cos\Delta\theta}\right) \\ &= \ddot{s}\frac{1 - \kappa_r d}{\cos\Delta\theta} + \frac{\dot{s^2}}{\cos\Delta\theta}\left[[1 - \kappa_r d]\tan\Delta\theta\,\Delta\theta' - [\kappa_r' d + \kappa_r d']\right] \end{aligned} \tag{3-15}$$

For high velocity driving scenario, derivative $\dot{d}$ and second derivative $\ddot{d}$ in time can be calculated:

$$\dot{d} = \frac{d}{dt} = \frac{\mathrm{d}s}{\mathrm{d}t}\frac{\mathrm{d}}{\mathrm{d}s}d = \dot{s}d \tag{3-16}$$

$$\ddot{d} = d'''\dot{s^2} + d'\ddot{s} \tag{3-17}$$

4 Framework Design

Based on the information from the upper layer the scenario recognition part will decide which the scenario the ego car currently in, and since the scenario has a significant influence on the behaviour logic of ego car, a series structure is applied here. The advantages of series structure, clear logic and reasonable planning, can guarantee the accuracy of scenario recognition and the depth of analysis to get the current scenario which helps to ensure the safety of the ego car. After determining the scenario, the behaviour decision layer will step into the state machine corresponding to the recognized scenario.

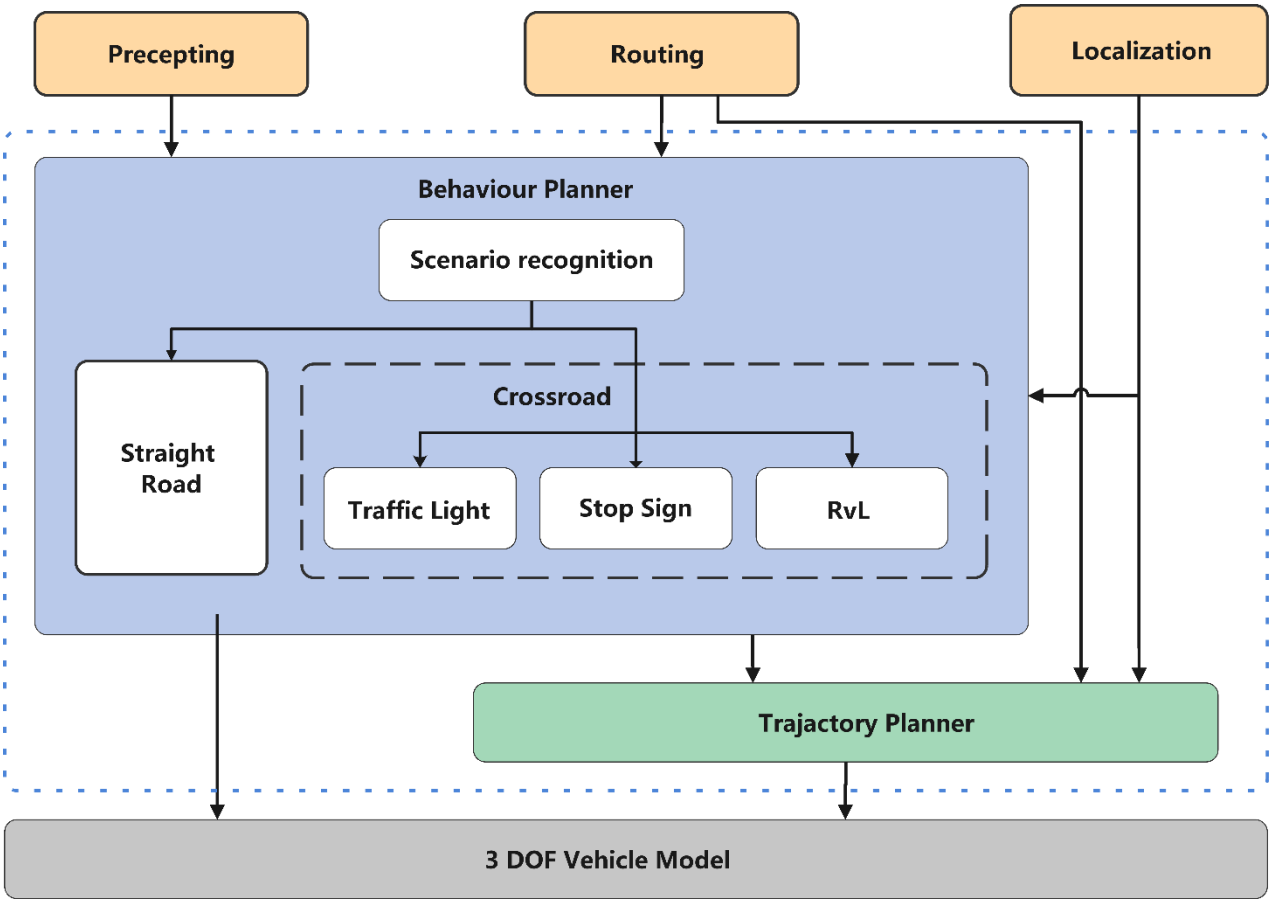


Figure 4-1: Modalized framework of urban autonomous driving

However, this project includes two scenarios such as straight driving and intersection, while the intersection scenario also is divided into three sub-scenarios, traffic light, stop sign and RvL. A series structured state machine may not manage such a large breadth of scenarios. So, the parallel structured state machine which capable with a larger number of scenarios is more suitable for such a situation after the scenario recognition. In the state machine, two parallel modules are straight driving and intersection scenario respectively. Besides, there are three parallel state machines under the intersection modules, which correspond to the traffic light, stop sign and RvL scenarios. Due to the modularity and expandability of the parallel structure, the finite state machine can easily add new scenarios

in later studies without affecting other existing modules. By using the hybrid structure of the finite state machine, the vehicle can not only reach a certain depth of analysis of its surroundings but also be able to deal with complicated various scenarios.

5 Behavior Planner Design

5.1 Finite-State Machine Class Design

According to different road conditions, the driving behaviors are divided into two scenarios, which are driving across crossing and driving along straight road. These two scenarios have something in common, for example both scenarios need collision check module and both of them should contain behaviors like turning right or left, keeping current lane, emergency stop etc.. But there are still differences between behaviors, as turning left in straight scenario actually means changing lane left and needs to consider vehicles running in the left lane, while turning left in crossing scenario implies turning to the left road, which must take right before left rule into account. Therefore, a unified code structure that can fully describe both the common and unique characteristics of scenarios is expected.

Finite state machine (FSM) is a powerful tool to implement reactive systems. It is composed of:

- States. Driving behaviors that are same of different in scenarios.
- Events. Conditions that enter or exist each
- Transitions. Conditions that need to be satisfied when transiting to next state.

We divide driving behaviors into followings parts:

Table 5-1: FSM response in traffic light scenario

Straight	Crossing
Common states	
Buffer	
Prepare Change Left (PCL)	
Prepare Change Right (PCR)	
Stop	
Unique states	
Keep Lane (KL)	Forwards
Lane Change Left (LCL)	Turn Right
Lane Change Right (LCR)	Turn Left

The simplest method to implement a Finite State machine is to use the procedural structure with multiple case statements. In this case, transition conditions of each state are expected to be defined separately. Once a new state that related with others appears, transition conditions of all other states need to be modified, which will make the whole system inflexible and hard to maintain.

Therefore, an object-oriented method is adopted so that the states can be dynamic linked with each other. In this method, each state is described as a class, transitions between states are designed as member functions of classes instead of pure case statements. Besides, to cover both common variables such as received messages from subscribed topics and distinct functions such as different transition conditions, a factory method design pattern is needed, where we consider the abstract structure of FSM state as the creator and concrete scenarios as concrete creators. Figure 4-1 shows an example from straight scenario. The `straight_fsm` is the abstract class that denotes the creator, and the rest 7 child classes are concrete creator that implement the creator.

Advantages of this kind of structure are shown as followings:

- Increasing the expandability and flexibility of the model, new scenarios can be easily added using the factory method.
- Encapsulating the process of state generation, which enhance cohesion of the code and reduce the coupling between parent and child classes.
- Easy for testing and maintenance.

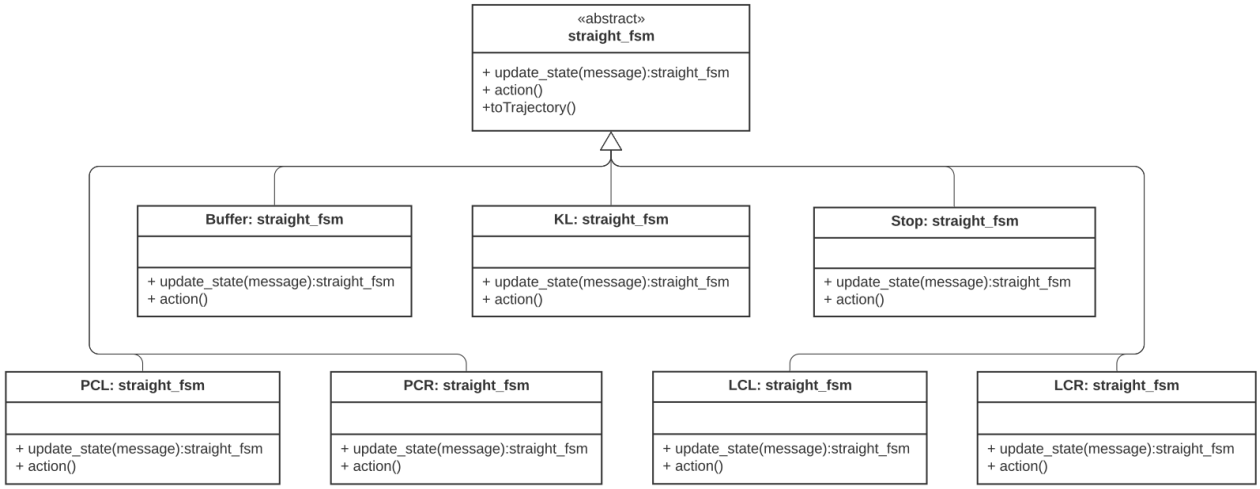


Figure 5-1: An example of class design of straight scenario.

5.2 Straight Behavior Planner

In this part, the deconstruction and analysis of an autonomous driving vehicle's behaviors for the straight driving scenario are implemented. Since the core of the project focuses on city driving, certain rules should be obeyed to drive behavior decision-making. A complete FSM generally consists of a few commonly used driving maneuvers. Four basic driving behavior modes are constructed for the straight driving scenario:

- Free driving mode: The ego vehicle keeps driving in the current lane, and there is no obstacle nearby.
- An automatic car following mode: The ego vehicle keeps driving with the same velocity as the current lane's front car.

- Lane changing mode: The ego vehicle will change lanes if it should drive in a different lane after the next intersection or overtake when the obstacles in front of the current lane are at a relatively slow speed.
- Emergency braking mode: The ego vehicle will stop immediately to avoid the collision with surrounding obstacles that change rapidly and dynamically.

According to the above analysis, a finite state machine is further designed based on a set of finite discrete states to solve behavior planning. One or more transitions can connect any states in a finite state machine, and sometimes one transition will return to the same state, which is called self-transition.

5.2.1 States for Straight FSM

Seven states are decided to represent all the possible situations that could happen in the straight driving scenario. Here is the states' form and instructions:

Table 5-2: State's form and instructions of straight FSM

State	Description
Buffer	Accepting command from the previous FSM or path planner || Transferring the current FSM information to the Trajectory planner and next FSM
Prepare Change Left / Right (PCL/PCR)	Activating the signal light and computing the possibility of turning right/left
Stop	Stop and requesting for a new Trajectory
Keep Lane (KL)	Driving at the current lane, including acceleration and deceleration
Lane Change Left (LCL)	Turning left/right

Some states will be introduced here with more details; moreover, all the calculations are based on the Frenet coordinates:

- Buffer: The ego vehicle will go to the buffer state when it begins a new straight FSM or finish a straight FSM.
- Keep lane: The ego car attempts to drive in the current lane by staying near the centerline of that lane, which also means that the direction d of the vehicle remains unchanged.
- Lane Change left/Right: As for the lane changes, the goal is to move from the original lane to the target lane. The target d is the lateral distance that the ego vehicle is expected to move at an appropriate time. For s, the same rules as keep lane state will be applied.
- Prep Change lane left/right: For security, different preparation states are added before the lane changes. In this state, the signal light will be turned on while lane changing feasibility is being analyzed. It also means that the ego car is still driving in the current lane with the same driving rules as the keep lane state.

5.2.2 Transitions for Straight FSM

To meet the vehicle autonomous driving requirement in a dynamic traffic environment, establishing a suitable driving behavior decision-making model is necessary. Therefore, the finite state machine's actual implementation needs to decide how these states are converted and what kind of inputs are used by the transition function.

Comprehensive situational information is the basis of states changing decision-making. The input information is composed of traffic information, environmental information, and vehicle status information. The input information is shown in Figure below.

In this scenario, decision-making is a further complicated procedure. Therefore, various traffic factors should be considered. The following diagram shows the transitions between all states.

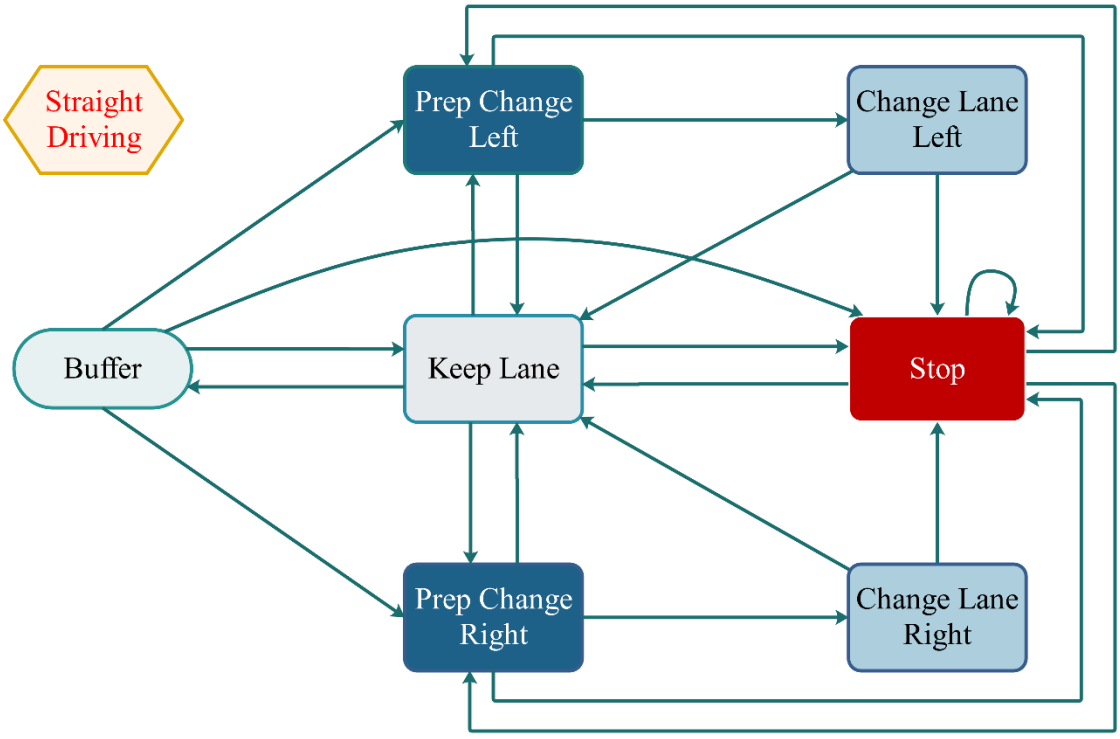


Figure 5-2: State transitions in straight driving FSM

However, the If-else decision tree is no more appropriate for some tricky situations, especially for the lane change. The cost function is therefore chosen to determine the factors that affect the lane changing decision.

5.2.2.1 Change Lane left/right

For the lane-changing behavior, it could be divided into mandatory lane changing and free lane changing. Decision-making of mandatory lane change is relatively simple, which could be generated based on the map information. It will be executed when the ego vehicle is expected to drive in a different lane after the next intersection (lane change right/left). However, the decision of free

lane change (lane change left) is more uncertain and random. It needs sufficient motivation and could be used as the conditions of state transition. After that, the probability of successfully changing lanes will be calculated.

In order to quantify the influence of different factors on ego vehicle decision-making, three penalty functions are established. All the penalty functions will be normalized, and the importance of each factor could be changed by adjusting the weights of the penalties.

The first one is the speed penalty function. For the sake of efficiency, the ego vehicle is expected to reach the destination as soon as possible. The overtaking will be considered If there is a relatively slow obstacle ahead in the current lane. The speed ratio function of these two cars is adopted here to express it.

$$cost_1 = \frac{v_{\text{obj}}}{v_{\text{ego}}} \tag{5-1}$$

where v_{obj} is the velocity of the obstacle in the front of the ego-car in the current road, and v_{ego} is the velocity of the ego car.

The second one is the penalty for the frequent lane changes. If the changed lane id is different from the target lane id at the next crossing, the ego vehicle should turn back to the original one after the overtake. A penalty is added here to avoid the frequent lane changes.

$$cost_2 = \begin{cases} 1 & id_{\text{target}} \neq id_{\text{change}} \\ 0 & id_{\text{target}} = id_{\text{change}} \end{cases} \tag{5-2}$$

Here, id_{target} refers to the id of the lane where ego-car is expected to drive after the next crossing, and id_{change} is the id of lane that ego-car will overtake.

The third is the distance penalty, which includes calculating two distances, the distance between the ego vehicle and the destination and the distance between the ego vehicle and the front obstacle. The first one is designed to reduce the possibility of overtaking when the ego vehicle arrives closer to the intersections. Furthermore, the ego vehicle will also not be allowed to overtake too early when it is far away from the front obstacle at the same time.

$$cost_3 = \frac{x_{\text{ego}} - x_{\text{start}}}{l_{\text{road}_i}} \tag{5-3}$$

$$cost_4 = \frac{x_{\text{obj}} - x_{ego}}{l_{\text{road}_i}} \tag{5-4}$$

where x_{ego} and x_{obj} are the s values of the ego-car and the front obstacle (the position in Frenet coordination), $x_{\text{start}}$is the start position of the i^{th} road and l_{road_i} is the whole length of this road.

Those penalty functions are combined to decide if the ego vehicle should overtake.

$$cost = w_1 cost_1 + w_2 cost_2 + w_3 cost_3 + w_4 cost_4 \tag{5-5}$$

When $cost$ is smaller than a criterium value, the ego car will execute overtake command. After that, enough driving space must be ensured for the successful implementation of the lane changing. Two "GAP" are measured here.

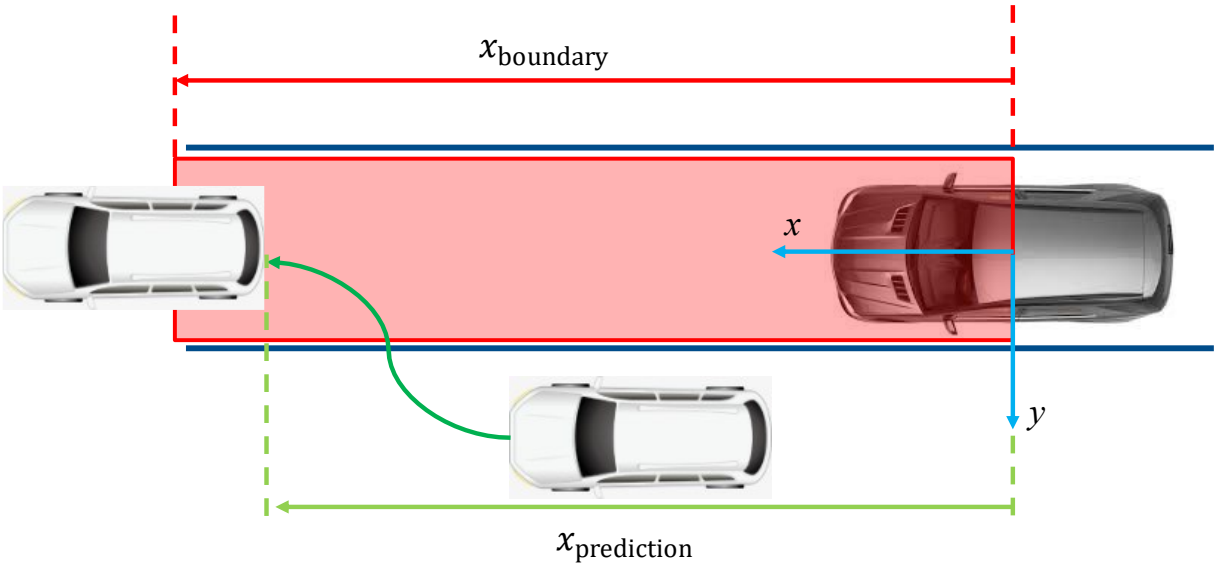


Figure 5-3: Boundary illustration

First, the surrounding vehicles in the target lane should remain enough space for the ego vehicle's entire body length.

$$x_{\text{boundary}} = (v_{\text{lf}} - v_{\text{lb}})T_{\text{change}} + (x_{\text{lf}} - x_{\text{lb}}) \tag{5-6}$$

Here, T_{change} is the time for car to change lane, $v_{\text{lf}}, v_{\text{lb}}$ are the velocity of the front/behind car in the left lane. x_{lf} and x_{lb} refer to the front car (tail) and the behind car (head) in the left lane.

Second, the minimum longitudinal distance between the ego vehicles and the other vehicles should also be large enough to avoid the collision when it cuts in.

$$x_{\text{prediction}} = \min\left(abs(x_{\text{lf}} - x_{ego}), abs(x_{\text{lb}} - x_{ego})\right) \tag{5-7}$$

5.3 Intersection Behavior Planner

As Traffic laws are different in every country, especially at intersections, it is not easy to handle the behavior using one algorithm. Concerning the Baidu Apollo autopilot framework[16], a series of finite state machines are excogitated based on specific scenarios to meet distinct safety requirements. The behavior state machine in Figure 5-4 exhibits a general structure representing the individual event responses and change states when necessary.

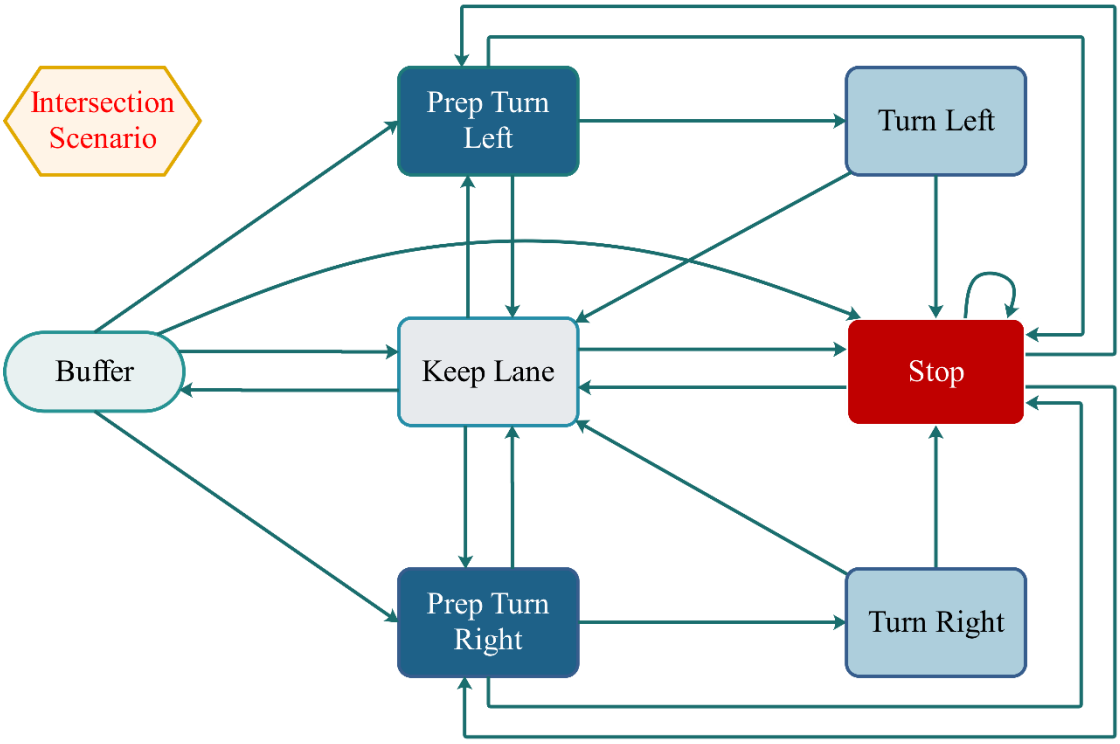


Figure 5-4: Intersection Behavior Planner FSM

Like the straight driving scenario, the following records all the used states in the FSM illustration and their associated accurate representations:

- Buffer
- Keep Lane
- Prep Turn Left/Right
- Turn Left/Right
- Stop

[16] Baidu: ApolloAuto/Apollo.

"Buffer" state represents the transition of the FSM in different scenarios. On the one hand, it could be the current FSM's initial state inherited from the last FSM. On the other hand, it could be the completed state of the current FSM, exported to other FSM in the next step.

As described by the FSM in the straight driving scenario, the "keep lane" state has two purposes in the intersection scenario. One is cruise control mode. In this mode, the ego car will maintain the city speed limit and drive along the established trajectory. The other is the vehicle following mode, which means that the ego car will maintain a certain relative distance from the vehicle detected in front, and the speed will be the same as the vehicle ahead. As the leader vehicle's speed changes, the ego car will adjust its speed and spacing accordingly.

From the functionality perspective, the "prepare turn left/right" state is the "keep lane" state's inheritance. The only difference is that prep will turn on the vehicle's blinker to communicate with other traffic participants. From the security view, the "turn left/right" state can only be achieved through the "prep turn left/right" state if the ego car locates an inside of an intersection. Appending prep state to FMS can reduce turning false triggers to a certain extent.

Stop state insists on keeping the minimum-security requirements. As in straight driving scenario defined collision check function, behavior planner in intersection scenario extends it to longitudinal and lateral obstacles in a wide range.

Since there are a large number of traffic participants in the intersection scenario, and different traffic laws need to be followed in the various scene, in the following section, the FSM strategy of the intersection scenario will be divided into three common sub-scenarios to introduce, which are sorted by road priority:

- Traffic light scenario
- Stop sign scenario
- "RvL" (in German "Rechts vor Links") scenario

5.3.1 Traffic Light Scenario

5.3.1.1 State decision in FSM

Among the intersection scenario, the most common scene with the highest road priority is the traffic light. In this scenario, all traffic participants must follow the established traffic rules: stop with a red light and go with a green light. Compared with the straight driving scenario, the traffic light scenario will be more complicated. On the one hand, it puts forward higher safety and legality requirements on FSM, namely: FSM needs to consider the dynamics of traffic participants in the current lane and recognize the traffic lights' instructions. On the other hand, FSM has new traffic efficiency requirements, which means that the vehicle cannot be too cautious when receiving traffic light signals. At the beginning of design, FSM must consider how to process traffic light information to meet the above requirements. Therefore, a new distance boundary $s_{\text{approach}}(v_x)$ was defined, which refers to the dynamic relative distance between the ego car and the traffic light intersection.

The relevant calculation is expressed in vehicle coordinate, which represents as follows:

$$s_{\text{approach}}(v_x) = (1 - p_{\text{emb}}) \cdot \frac{v_x^2}{a_{\text{comfort}}(v_x)} + p_{\text{emb}} \cdot \frac{v_x^2}{a_{\text{emb}}} + v_x \cdot t_f + s_{\text{emb}} \tag{5-8}$$

where p_{emb} indicates the weight of emergence braking, v_x signifies the ego car longitudinal velocity, $a_{\text{comfort}}(v_x)$ means the comfort approaching deceleration, a_{emb} denotes the emergency deceleration, t_f is the FSM update duration, and s_{emb} intends the minimal emergency braking distance in low velocity situation.

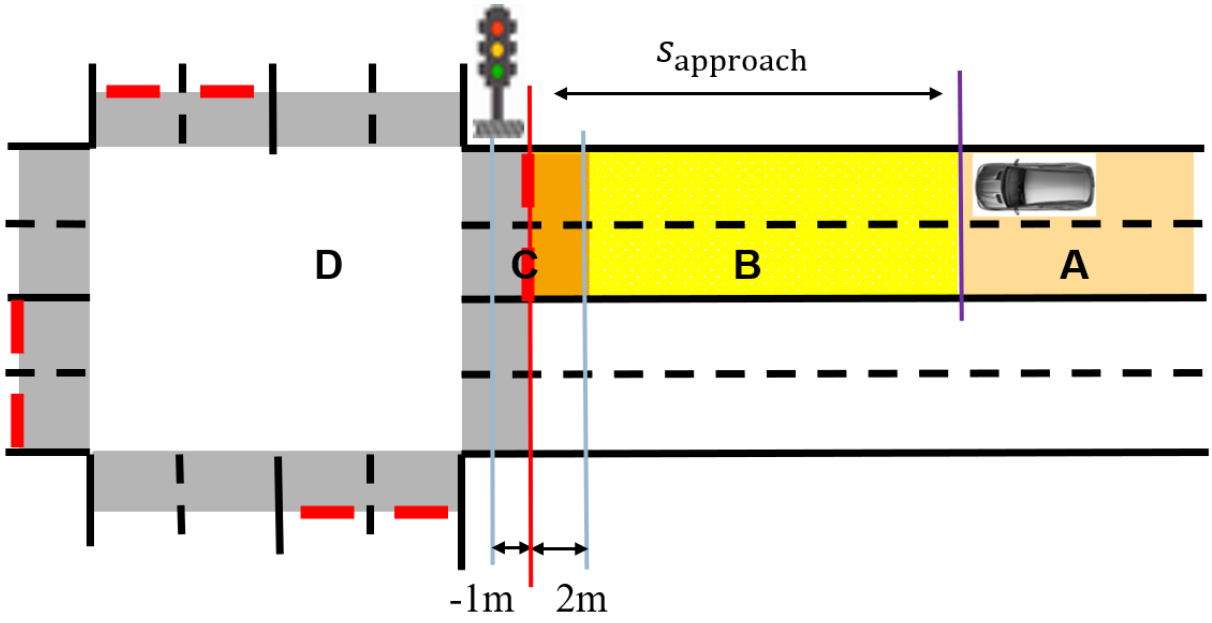


Figure 5-5: intersection area division in traffic light scenario

The above Figure 5-5 shows that the relative distance between the ego car and traffic light divides the actual traffic lane into four areas. Once the ego car passes this boundary, the traffic light's indication starts to affect the ego car state. For example, in area B, when the traffic light is red or turns yellow from green, the ego car will start to break; when the traffic light is green or turns yellow from red, the vehicle is ready to pass through the intersection. The following table enumerates the FSM's response to the traffic light in each area.

Table 5-3: FSM response in traffic light scenario

Area	Description	Response
A	Far away from intersection	Ego will keep forwards, no response to traffic light.
B	Approaching intersection	Ego car starts to brake if traffic light is red or turn yellow from green.
C	Crosswalk	Ego car starts to brake if traffic light is red.
D	Inside of intersection	Ego car will go along the planned trajectory, no response to traffic light.

The above is the description of the distance calculation process. In order to clearly distinguish the influence of various factors in the state transition of FSM, the above area division is presented as a function in the code separately.

The following flowchart clearly illustrates the workflow of `TrafficLightCondition::area_recognition`.

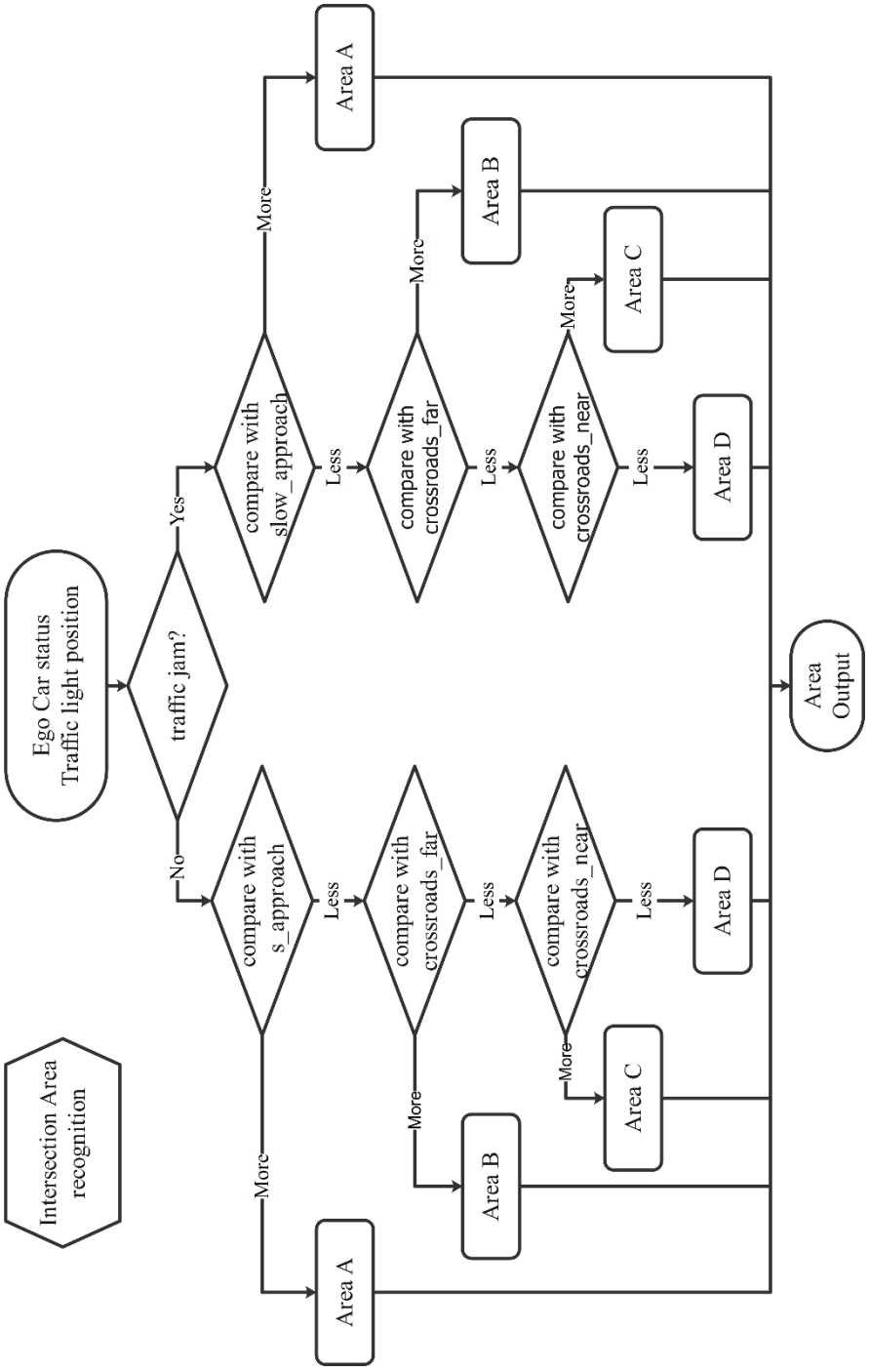


Figure 5-6: Intersection area recognition

At the beginning of the design, in order to distinguish between high-speed and low-speed functionality, the function was specifically differentiated based on the low-velocity threshold. The distance between the leader vehicle and the ego car in a traffic jam can be independently set to achieve higher traffic efficiency through the threshold. The following table lists the parameters that have appeared in this function.

Table 5-4: Parameters in area recognition

Parameter	Value	Description
v_slow	2	velocity threshold for traffic jam or creep driving
slow_approach	7	traffic jam distance, threshold to start the braking
s approach	$s_{\text{approach}}(v_x)$	dynamic relative traffic light distance
crosswalk_pose_x_far	2	crosswalk far offset relevant to traffic light stop line
crosswalk_pose_x_near	-1	crosswalk near offset relevant to traffic light stop line

Moreover, FSM needs to consider other traffic participants on the road and respond to obstacles at different distances and relative speeds. For example, if obstacles are too close to the ego car or the relative velocity is too high, it will cause the ego car to break. In FSM, the longitudinal comfort braking boundary $s_{\text{comfort}}(v_x)$ is proposed for consistent and general evaluation criteria, as expressed by the following formula.

$$s_{\text{comfort}}(v_x) = \frac{\Delta v_x^2}{a_{\text{comfort}}(v_x)} + v_x \cdot t_f + s_{\text{emb}} + \frac{x_{\text{ego,body}}}{2} \tag{5-9}$$

where Δv_x^2 indicates the longitudinal relative velocity between ego car and the obstacle in vehicle coordinate, and $x_{\text{ego,body}}$ signifies the ego car bodywork length, which aims to compensate for the effect of coordinate offset.

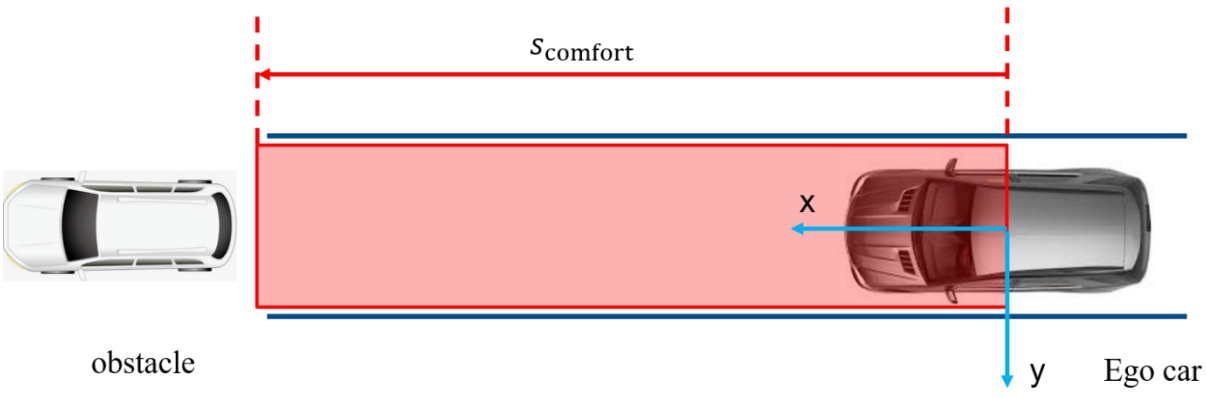


Figure 5-7: Comfort braking boundary in vehicle coordinate

Figure 5-7 above exhibits an example under vehicle coordinate. Once the obstacles are located in the boundary box, the ego car will automatically recognize the risk of collision ahead.

As mentioned at the beginning of this section, FSM needs to analyze transverse oncoming traffic in the intersection area. From the safety perspective, all traffic participants inside of intersection can be classified into three main groups in a generalized traffic light scenario[17]:

- Protected: the ego car must pass through an intersection with a clear traffic light indicator—a significant arrow in green for the corresponding turn.
- Unprotected Left: the ego car will make a left turn without a distinguished light, which intends that the ego car would need to wait for the oncoming vehicle until free.
- Unprotected Right: The ego car would turn right during a possible oncoming vehicle.

Therefore, FSM expands the cut in collision detection based on straight driving scenarios to reduce transverse collision risk. In the area approaching the intersection, FSM will process the scanned obstacles to observe whether there is a possibility of collision.

$$s_{\text{transverse},x} = \Delta s_x + \Delta v_x \cdot t_{x,\text{cut in}} \tag{5-10}$$

$$s_{\text{transverse},y} = \Delta s_y + \Delta v_y \cdot t_{y,\text{cut in}} \tag{5-11}$$

where $t_{x,\text{cut in}} = \frac{\Delta s_y}{\Delta v_y}$ denotes the predicted longitudinal cut-in duration, $t_{y,\text{cut in}} = \frac{\Delta s_x}{\Delta v_x}$ indicates the predicted lateral cut-in duration, Δs_x means the longitudinal relative distance in vehicle coordinate, Δs_y represents the corresponding lateral distance.

The illustration shows an extreme collision case below.

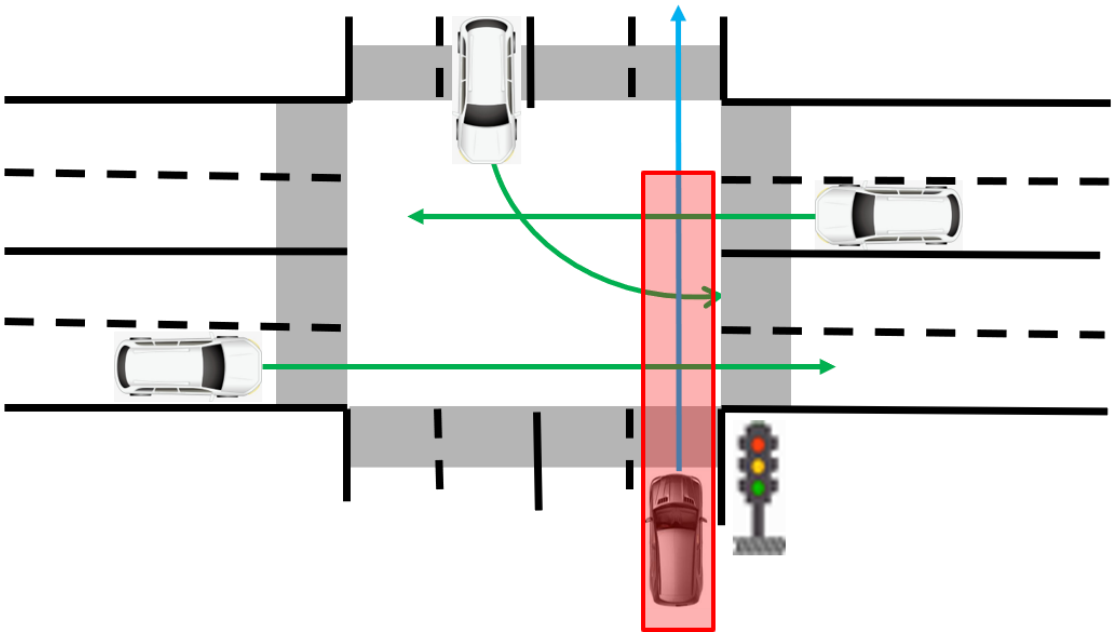


Figure 5-8: Oncoming vehicles collision inside the intersection

[17] Baidu: ApolloAuto/Apollo.

The current safety detection of FSM in the intersection scenario is based on the vehicle coordinate, which means that the longitudinal safety distance calculation and lateral collision prediction can only be suitable for straight driving obstacles. In vehicle coordinate, the prediction of cut-in duration $t_{\text{cut in},x}$ and $t_{\text{cut in},y}$ are no longer accurate for turning oncoming vehicles. Besides, when the ego car turns left or right, transverse collision prediction may involve irrelevant obstacles, which leads to unnecessary braking instructions in FSM. In response to this problem, the relevant improvement measure is to process all collision predictions based on the Frenet coordinate system. However, the frenetic coordinate conversion requires polynomial interpolation and matrix inverse operations, which will bring additional computational pressure in the real-time FSM. In the relevant code, the interface for Frenet coordinate conversion has been reserved, which can be improved in subsequent projects. Due to the time limit of the project process, the transverse collision prediction is always applied to the vehicle coordinate.

Longitudinal safety distance calculation and transverse collision prediction are the core parts of safety detection. According to principle of least knowledge[18], safety detection exists as an independent function `TrafficLightCondition::collision_check`, which is represented in the Figure 5-9 below.

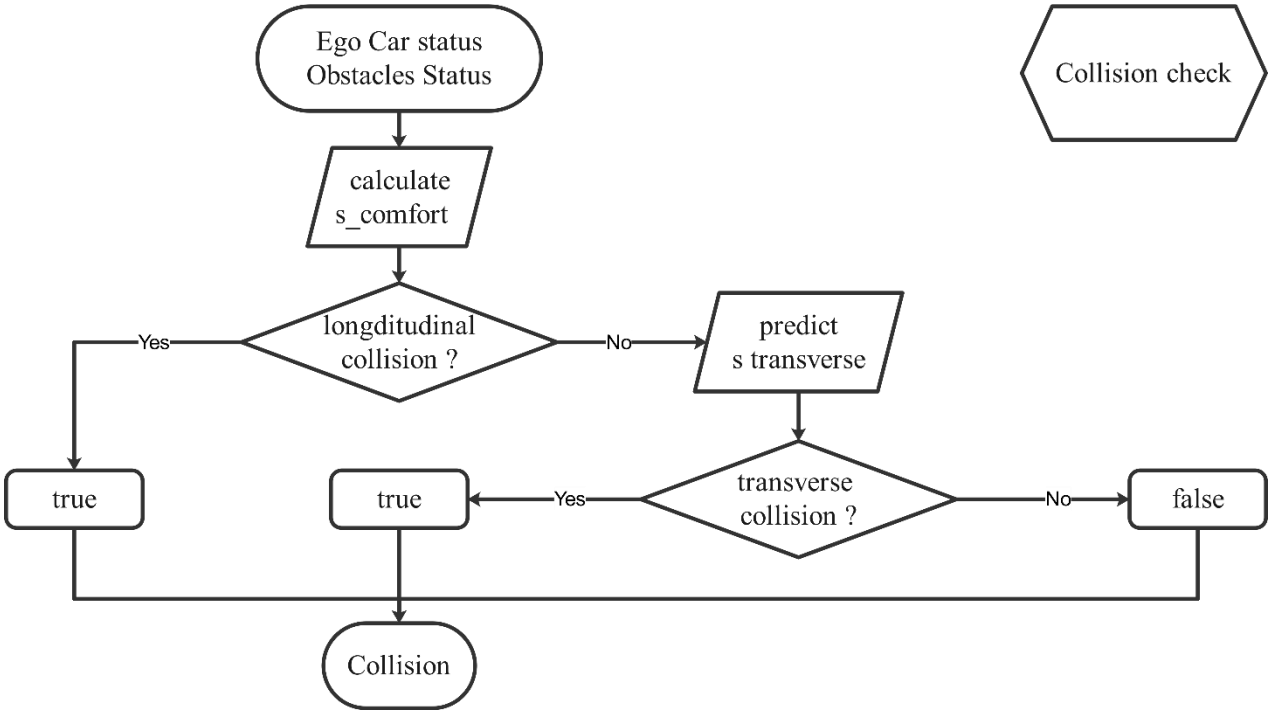


Figure 5-9: Collision check flowchart

In addition to collision check, in the traffic light FSM state transition class `TrafficLightCondition`, the function `TrafficLightCondition:: motivation_detection`, which is based on

[18] K. J. Lieberherr; I. M. Holland: Assuring Good Style for Object-Oriented Programs (1989).

global planning, and the function `TrafficLightCondition:: yellow_check`, which depends on the traffic light's basic signal, are also defined.

As the name suggest, `TrafficLightCondition:: motivation_detection` 's role is to determine the ego car's motion motivation in the current intersection, such as going straight, turning left, or turning right. For the current project, due to the lack of an effective global planning module, motion motivation can only be entered manually. Due to the lack of a practical global planning module, the motion motivation can only be entered manually for the current project.

Based on the sequence of the red and the green light, `TrafficLightCondition:: yellow_check` determines the yellow light conversion process. But in CarMaker's virtual environment, the simulator can directly send out a yellow light signal and indicate the signal conversion. Therefore, `Traffic-LightCondition:: yellow_check` is only used as a functional backup during actual vehicle testing. Here list the independent functional functions, which are callable in the state transition of traffic light scenario:

- `int TrafficLightCondition::motivation_detection() {…}`
- `bool TrafficLightCondition::collision_check() {…}`
- `int TrafficLightCondition::yellow_recognition() {…}`
- `int TrafficLightCondition::area_recognition() {…}`

Consequently, based on the return values, FSM can make appropriate state transitions by combining the functions mentioned above. The illustration below explicates the function-calling process of state transition considerably.

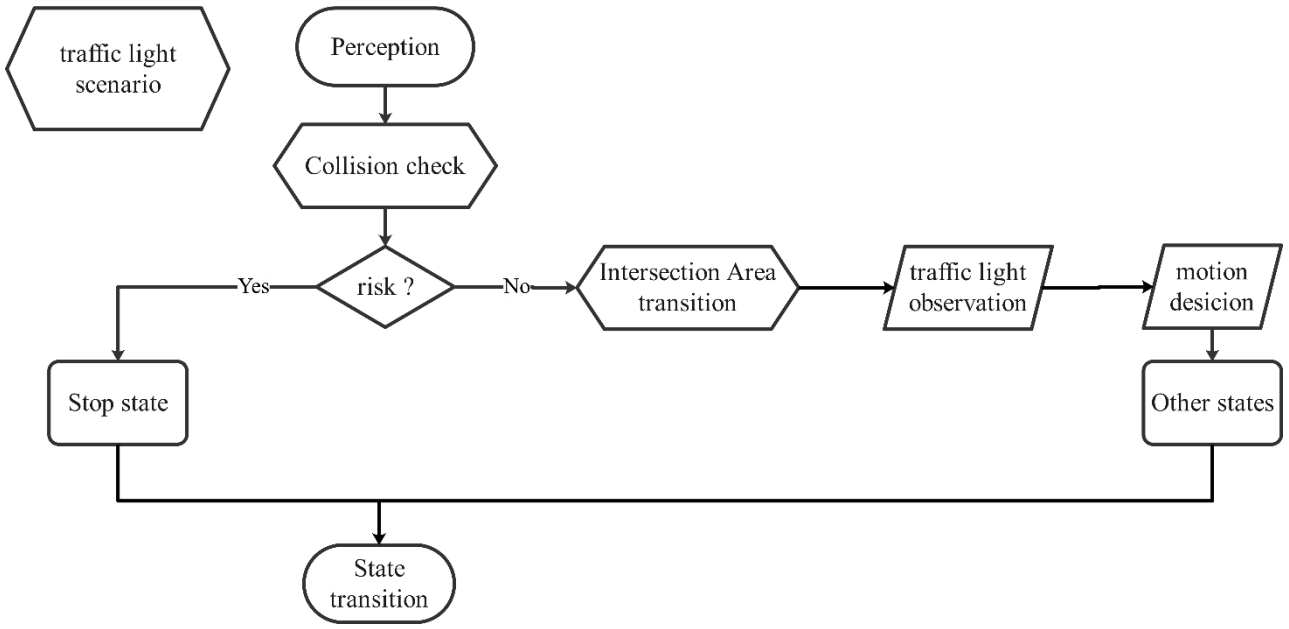


Figure 5-10: State transition process

From a safety perspective, the first considered function should be the collision check, which is used to determine whether there is a possibility of a collision in the current vehicle coordinate system. Secondly, the intersection area will be called because it is based on processing motion and traffic light information. After that, by considering motion and signal lights, FSM will determine which state to choose. All transition states are listed as follows:

- `bool TrafficLightCondition::isBufferToKeepLane() {…}`
- `bool TrafficLightCondition::isBufferToPrepLeft() {…}`
- `bool TrafficLightCondition::isBufferToPrepRight() {…}`
- `bool TrafficLightCondition::isBufferToStop() {…}`
- `bool TrafficLightCondition::isKeepLaneReflexive() {…}`
- `bool TrafficLightCondition::isKeepLaneToPrepLeft() {…}`
- `bool TrafficLightCondition::isKeepLaneToPrepRight() {…}`
- `bool TrafficLightCondition::isKeepLaneToStop() {…}`
- `bool TrafficLightCondition::isKeepLaneToBuffer() {…}`
- `bool TrafficLightCondition::isStopReflexive() {…}`
- `bool TrafficLightCondition::isStopToKeepLane() {…}`
- `bool TrafficLightCondition::isStopToPrepLeft() {…}`
- `bool TrafficLightCondition::isStopToPrepRight() {…}`
- `bool TrafficLightCondition::isPrepLeftToKeepLane() {…}`
- `bool TrafficLightCondition::isPrepLeftToStop() {…}`
- `bool TrafficLightCondition::isPrepLeftToTurnLeft() {…}`
- `bool TrafficLightCondition::isPrepRightToKeepLane() {…}`
- `bool TrafficLightCondition::isPrepRightToStop() {…}`
- `bool TrafficLightCondition::isPrepRightToTurnRight() {…}`
- `bool TrafficLightCondition::isTurnLeftToKeepLane() {…}`
- `bool TrafficLightCondition::isTurnLeftToStop() {…}`
- `bool TrafficLightCondition::isTurnRightToKeepLane() {…}`
- `bool TrafficLightCondition::isTurnRightToStop() {…}`

A detailed description of each transition condition can be obtained in the gitlab libraray, and the tedious repetition will not be repeated here.

So far, the state processing of FSM has been completed. The next section will focus on the output of FSM, namely, the predicated motion target.

5.3.1.2 Motion target prediction

In the decision-making layer, the behavior planner will not only output the current state but also predict a motion target, which contains the position, velocity, and acceleration. In order to balance safety and efficiency, the choice of motion target depends on three conditions:

- The current dynamic status of obstacles
- The current dynamic status of ego car
- The current FSM state

Theoretically, the features contained in the motion target, such as the displacement, velocity, and acceleration, should all be considered in the Frenet coordinate system. However, in practice, due to the influence of map accuracy, the frenetic conversion may not be accurate. Since it involves matrix inversion and spline interpolation, it could be treated as an algorithm optimization challenge for real-time systems. Limited by the laptop's computing power, in this project, the prediction of the motion target will be performed under the vehicle coordinate system.

As mentioned before, the motion target contains three levels of features. First of all, its prediction will start from the simplest acceleration level. From the comfort perspective, the acceleration of the motion target should be zero. It can avoid jerk from the trajectory generation level and simplify the calculation of trajectory generation.

The next content to be discussed is the speed selection of the motion target. From a security perspective, obstacle' influence needs to be considered firstly. If there is an obstacle in front of the vehicle, the speed of the motion target should be set the same as the speed of the current obstacle. Of course, there are exceptions. If the speed of obstacles exceeds the road speed limit, the motion target's speed should be selected as the local speed limit.

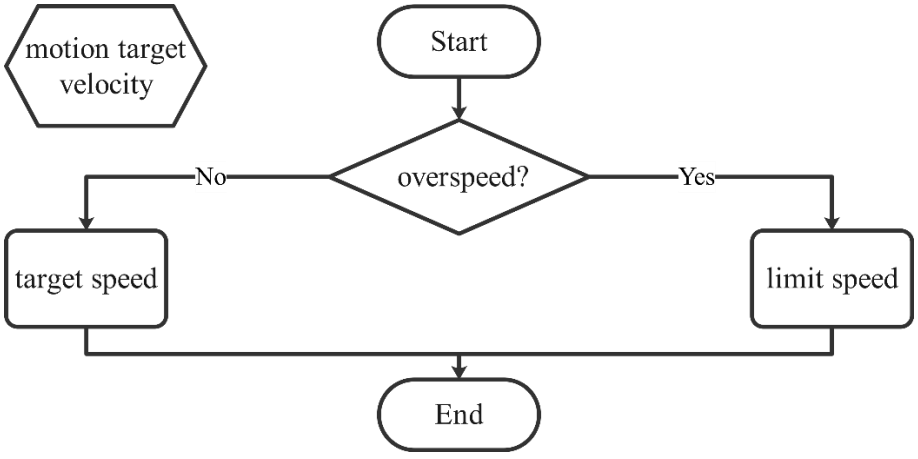


Figure 5-11: Motion target velocity selection

The image above shows the speed selection process. Although this inherited method is relatively crude, it is feasible in most straightforward situations. Combined with selecting appropriate obstacles,

the speed selection of the motion target can quickly realize the function of cruise control. For curved sections, due to the limitation of the radar detection angle, in the intersection section, the ego car cannot detect far objects in the normal direction, which reduces the mistake caused by the directly inherited speed.

Finally, the motion target needs to predict a relative displacement to indicate where the ego car should reach after a period of driving. The selection of this displacement is essential, which means that the inappropriate displacement may cause the failure of trajectory generation. The following content will focus on the estimation of this relative displacement.

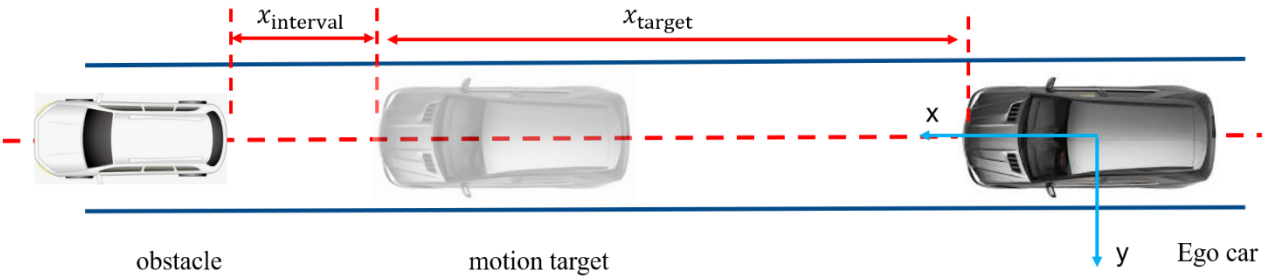


Figure 5-12: Displacement prediction in straight driving under vehicle coordinate

Above Figure 5-12 exhibits the estimation of displacement under intuitive conditions: straight driving with detected obstacle ahead. Two parameters need to be introduced here, namely the estimated time t_{motion} and the displacement interval x_{interval}. The estimated time is inspired by the concept of TTC: Under a given time constant, the travel distance of ego car depends on its velocity. Besides, the displacement interval is proposed to avoid collisions. In this case, the estimated longitudinal relative displacement needs to minus the displacement interval for safety requirements. The mathematical expression is shown as follows:

$$x_{\text{target}} = \min\left(\frac{v_x + v_{x,\text{target}}}{2} \cdot t_{\text{motion}}, x_{\text{obstacle}}\right) - x_{\text{interval}} \tag{5-12}$$

$$y_{\text{target}} = \min\left(\frac{v_y + v_{y,\text{target}}}{2} \cdot t_{\text{motion}}, y_{\text{obstacle}}\right) \tag{5-13}$$

where v_y represents the lateral velocity of ego bar, $v_{x,\text{target}}$ indicates the longitudinal velocity of target motion, $v_{y,\text{target}}$ denotes the lateral velocity of target motion, x_{obstacle} indicates the longitudinal relative distance of the detected obstacle, and y_{obstacle} denotes the lateral relative distance of the detected obstacle.

If there are no obstacles in front of the vehicle, the above formula can be simplified to the following expression:

$$x_{\text{target}} = v_x \cdot t_{\text{motion}} \tag{5-14}$$

$$y_{\text{target}} = v_y \cdot t_{\text{motion}} \tag{5-15}$$

However, those expression methods also have its own limitations, which is shown as below.

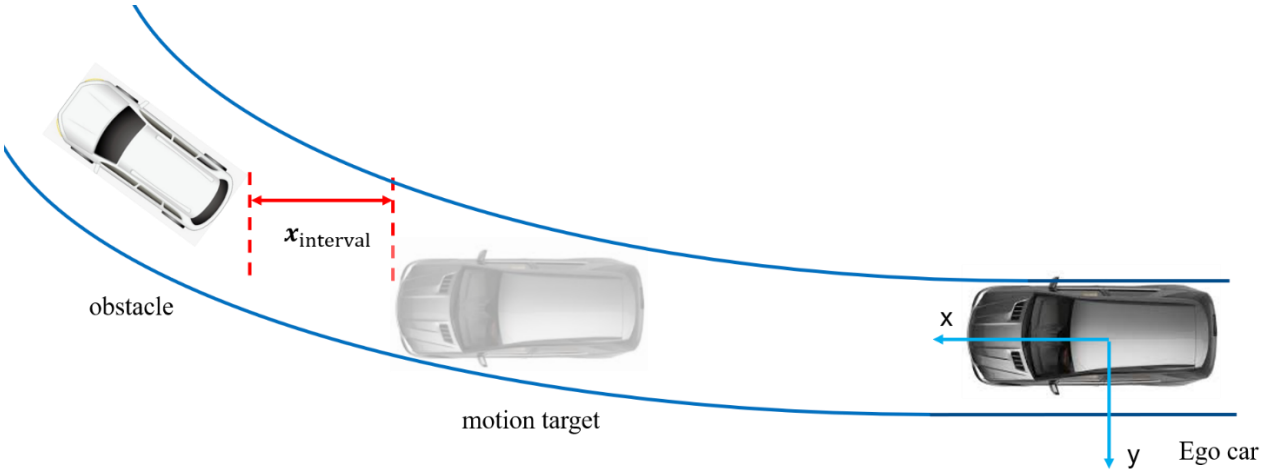


Figure 5-13: Displacement prediction in curve driving under vehicle coordinate

Figure 5-13 shows an extreme case that is not associated with relative displacement estimation. When obstacles are stationary at the curve's entrance, the estimation of the lateral direction will not be accurate. In this case, the estimated motion target will deviate from the centerline of the path lane. Therefore, the trajectory planner must preprocess the motion target: execute frenetic conversion on the motion target. The expression of the motion target in the frenetic coordinate system can be obtained by conversion. If the FSM state is not a change lane, the normal offset can be set to 0 to eliminate the influence of this kind of motion target error estimation.

Before the motion target is passed to the trajectory planner, it is necessary to check the relative displacement validity. In the above calculation process, the interval displacement value is constant. Therefore, there is a risk that the longitudinal displacement of the motion target may be a negative value, especially in traffic jams.

The following Figure 14 shows the entire process of motion target displacement estimation.

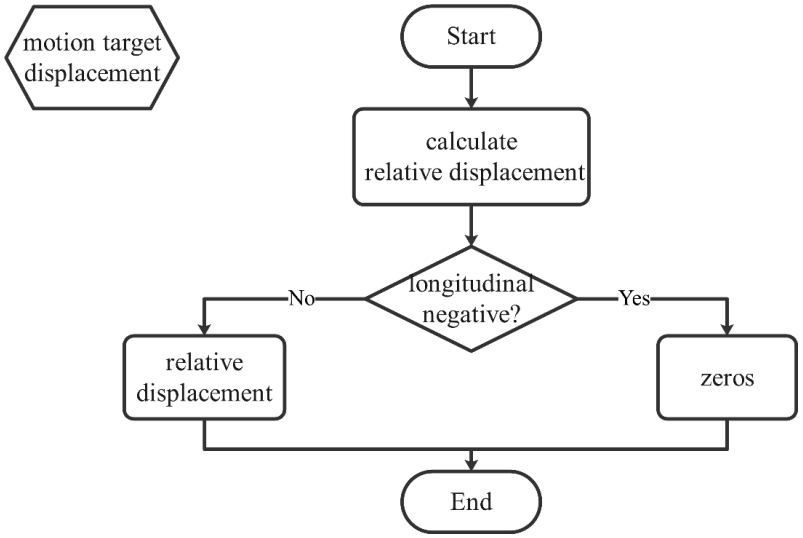


Figure 5-14: Motion target prediction process

At the end of this section, the following table lists the constants involved in all motion target prediction.

Table 5-5: Parameters explanation of motion target prediction

Parameter	Value	Unit	Description
x_{interval}	4	m	displacement interval, as offset in longitudinal direction
t_{motion}	2.4	s	estimated time constant for motion target displacement

5.3.2 Stop Sign Scenario

In the stop sign scenario, the behavior logic is similar to the traffic light scenario, therefore, the area recognition function and the collision check function make no difference. Moreover, the behavior, which the ego car waits at the intersection until it passes through the intersection, is related to the traffic sign and the detected vehicles in other lanes. In this scenario, before crossing the intersection, the ego car must stop within the stop area and wait until none of the interfered cars or pedestrians appears in interested areas.

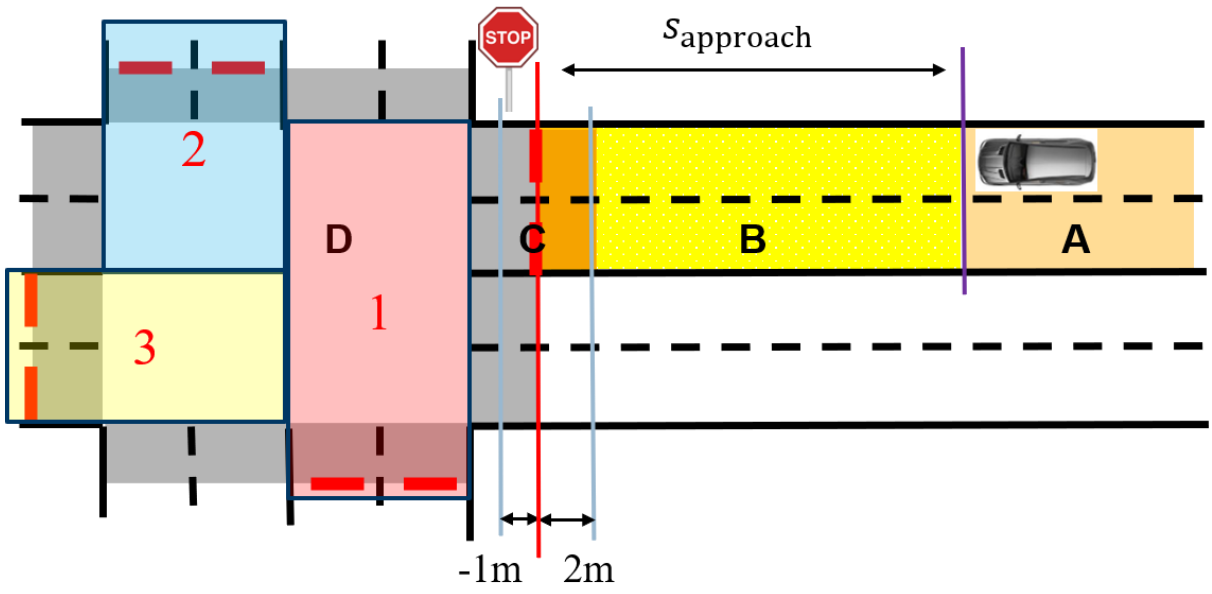


Figure 5-15: Intersection area division in stop scenario

Furthermore, in order to analyze whether the oncoming vehicle from street does interfere with the ego car, detection areas have been defined for other three street in the design of transition conditions. In the Figure 5-15 above, three detection areas are divided by different directions at the intersection, which are listed in the following table.

Table 5-6: Parameters in area recognition

Area	Priority	Description
detection area 1	top	left oncoming car street travels straight through the intersection
detection area 2	medium	right oncoming car street travels straight through intersection
detection area 3	low	opposite oncoming car turns left through intersection

When oncoming vehicles travel in detection areas, they could pose a serious threat to security of the ego car. In this case, the ego car must continue to wait at the stop line; only if all detection areas are clear, the vehicle could restart to travel through the intersection. The flow chart below clearly illustrates above logic.

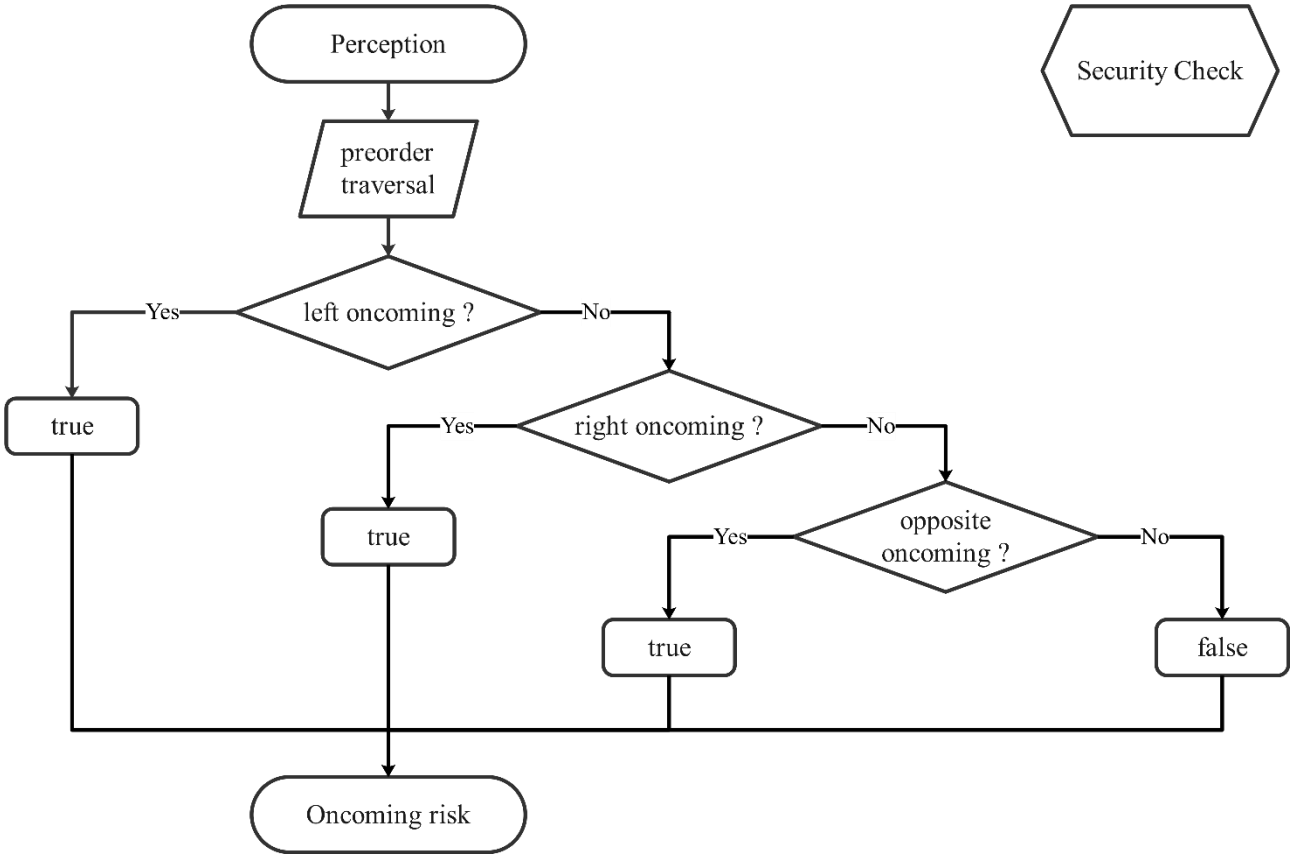


Figure 5-16: Security check in stop sign scenario

In the implementation of `security_check()` function, according to the three detection areas it is divided into three sub-functions `check_left()`, `check_right()`, `check_right()`.The following table list the relevant parameters in the function

Table 5-7: Parameters explanation of check right

Parameter	Description
Right_boundary_x_static	The detection area boundary in the intersection
Boundary_y_near_dynamic	Coefficient to form the dynamic boundary
Enitity_position_x	Vehicle position relative to stop line
Object_velocity_in_stopline	Vehicle velocity relative to stop line
Area_security	If vehicles exist in the detection area

The variables in this function include the position of the stop sign, the position and velocity of the detected vehicles near the intersection. These data are available in the `queryVar` Struct variable in the class `StopSignCondition`. Firstly, the state machine selects the vehicles driving into the intersection from the message `DetectedObjectArray` based on their position in the global coordinate system, while categorizing them into three different directions. After that, the relative distance between the vehicle and the stop line is calculated in order to figure out whether the vehicle is in the

detection area. The following illustrates the workflow of `check_right()`. The other two function has the same structure.

The speed of the oncoming vehicle is also an important factor in dividing these detection zones. In a hypothetical situation, the ego car does not comply with traffic regulations, i.e., it does not wait at the stop sign until the surrounding area is safe enough to drive through. In such scenario, the oncoming vehicle needs to take evasive action to avoid the collision. Based on the above consideration, the length of the detection area should be a function of the oncoming vehicle's velocity, which is in the stop line coordinate system. The original point of this coordinate system is middle point of the stop line and the x-axis points to the direction leaves the intersection. In the implementation of the function, vehicle's velocity also needs to be transformed from the global coordinate system to the stop line coordinate system. This dynamic boundary line is calculated by multiplying oncoming vehicle's velocity by the static variable `boundary_y_near_dynamic` in the class `StopSignCondition`.

On the opposite side of dynamic boundary, the detection area should reach into the intersection, which is under the consideration of the surrogate safety indicators PET. The width of the detection area is half of the entire road width, assuming that there are an equal number of lanes in both directions to the intersection along the road.

In this function each of the three detection areas will be checked for safety and returns a Boolean variable, which these are packaged in a Struct variable `Security` as the return value of the function `security_check()`.

Besides, a function `queue_check()` checks whether the ego car is the first car in front of the stop line, that is, no other obstacle and vehicles between the ego car and the stop line. if not, this function will return a true value. Only when the ego car is the first car in the queue, it will start detecting if an oncoming car or obstacle in the detection area. The function `stop_times_counter()` records the time that the vehicle has stopped due to the stop line at the intersection. The following two tables list the parameter used in the function and also the two below is about the workflow of these two functions.

Table 5-8: Parameters explanation of queue check

Parameter	Description
Traffic_rel_pos	Distance between ego car and stop sign
Crosswalk_pose_x_far	Crosswalk boundary near the egocar side

Table 5-9: Parameters explanation of stop times counter

Parameter	Description
First_queue	If the ego car is the first car at queue
Stop_catch	Total time the ego car stops at stop sign

In the stop sign scenario, the structure of the finite state machine including the ego car's states are also the same as the traffic light scenario but differs in transition conditions, because different traffic

sign rules the vehicle behavior in a different way. In transition conditions it is not necessary to consider the state of the traffic light, but the lane area recognition, motivation of the ego car, collision check and the state of the detection area are still important factors. The sequence of the ego car in the queue and the time which ego car has stopped at the stop sign are also the variables of the transition condition.

5.3.3 "RvL" Scenario

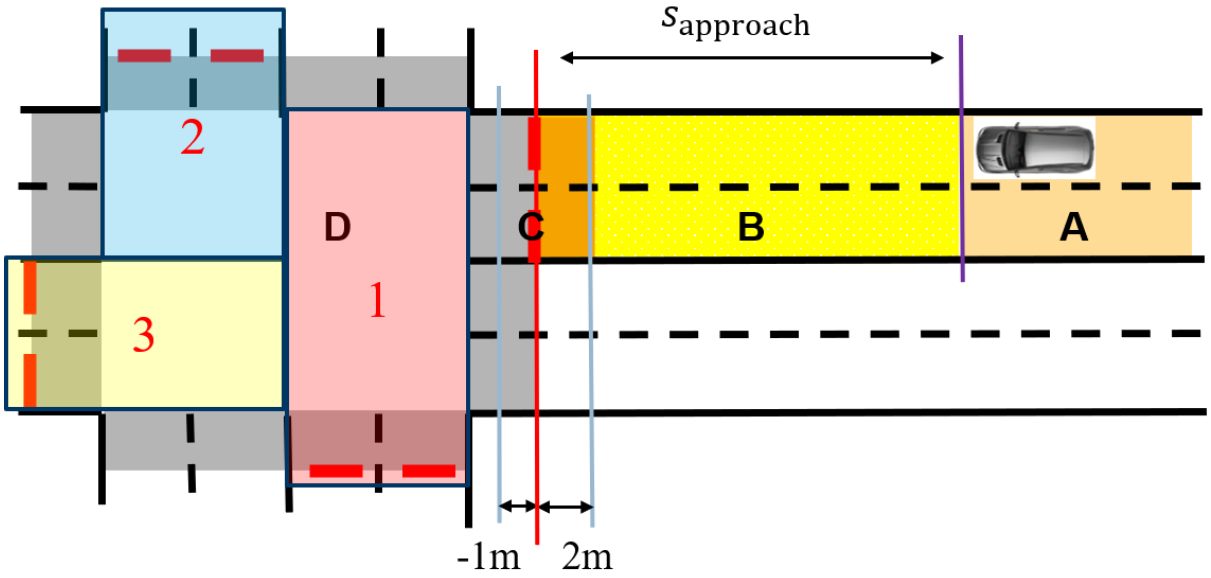


Figure 5-17: Intersection area division in RvL scenario

In the RvL scenario, since there are no traffic signs at the intersection to control the behavior of the vehicles, the ego car can travel directly through the intersection when there are no vehicles in the detection area. This function is also implemented in `security_check()` as applied in the stop sign scenario. The RvL scenario is also validated by the finite state machine with the same structure and state as in the above two scenarios. However, compared to the previous two scenarios, the variables in the transfer condition are only lane zone recognition, vehicle motive, collision detection and state of detection zones.

6 Trajectory Planner Design

In the process of trajectory generation, an optimal control approach will be applied to the project. Unlike other optimization problems, the key point is not minimizing or maximizing a specific cost function, but searching for a strategy, which ensures that an optimal solution is retained and once it can be found. Rather than trying to solve the problem using the optimal algorithm under a specific cost function, the multiple trajectories are generated by the sampling data and the start/end states, which the behavior planner gives. Meanwhile, the cost function is used to evaluate whether those trajectories are beneficial. The trajectory with the lowest value will be the best.

It is comprehended that Jerk plays an essential role in evaluating the trajectories. According to Takahashi (1989)[19], in a one-dimensional problem the quintic-polynomials are the jerk-optimal connection between a start state $Q_0 = [q_0, \dot{q_0}, \ddot{q_0}]$ and an end state $Q_1 = [q_1, \dot{q_1}, \ddot{q_1}]$ within the time interval $t_f := t_1 - t_0$. With all the mentioned data, a cost function can be constructed for the project. Because the quintic-polynomials are jerk-optimal, they should also be applied to generating the trajectories. Figure 6-1 shows the process of optimal trajectory generation.

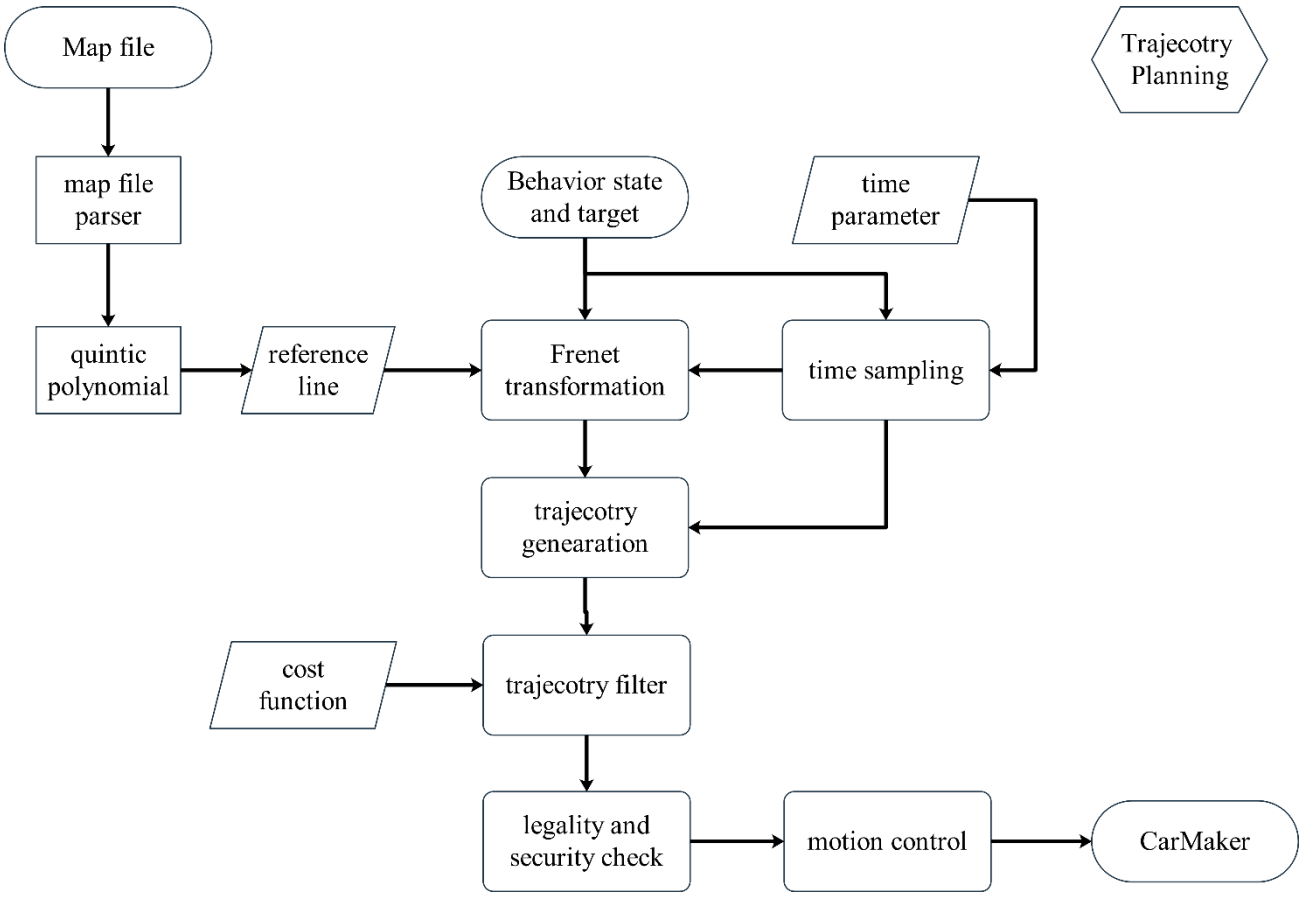


Figure 6-1: Trajectory generation

[19] A. Takahashi et al.: Local Path Planning And Motion Control For Agv In Positioning (1989).

First, the trajectory planner receives the order and necessary data from the behavior planner. Then the trajectory planner generates multiple trajectories with the data. After multiple trajectories are generated, the security check will be applied to the trajectory group. The cost function will rank only the trajectories which pass the security check. Finally, the trajectory with the lowest value is exported as the best trajectory. In the next pages, the details of the trajectory generator and the cost function design will be introduced.

6.1 Map Parser (Waypoint loader)

6.1.1 Introduction to OpenDRIVE Format

OpenDRIVE is an XML-based high precision map format, designed for auto navigation and receives worldwide recognition in automated driving simulation field. OpenDRIVE map defines a hierarchical structure, which contain various geometry description of road network, such as road marking, traffic signs road profiles and so on.

In our project, necessary simplification is adapted to accelerate the map parsing process, like no evaluation profile of the road surface. The output of the map parsing process is a data structure called Waypoint, which contains an array of cartesian coordinate of the required route's reference line. This route should be generated by global path planner in advance. Under this case, only two types elements would be extracted: reference line and lane relevant description. The whole map would be treated as a 2-D space to simply the calculation.

6.1.2 XML Parser

Extensible Markup Language (XML) plays an important role in the OpenDRIVE map format. Query, reading and writing of the stored date are based on the XML parser to access different elements and its attributes. There are many opensource third-party libraries for the XML language parsing on different Platform. In our case, two XML parser would be selected: by C++ tinyXML2 and by Python Beautifulsoup4. These two libraries are beginner friendly and also lightweight, which could be flexibly and easily integrated into our code project.

Python Parser BeautifulSoup4 is used for the early-stage development. With Python, the coding parted could be extraordinarily simplified, especially when it comes to great amount of array or vector. Meanwhile the C++ version of tinyxml2 serves as backup for the further development, which could meet the demanding performance of accessing the data by real-time navigation.

6.1.3 Geometry Information Extraction

The key idea of the parsing is generating array list of reference line of each road in the predefined route, which are described in cartesian coordinate. A simplified tree of the OpenDRIVE, only with the desired lane and reference line geometry information, is shown as below:

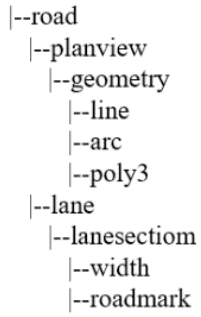


Figure 6-2: Simplified tree structure of the OpenDRIVE children node

As the Figure 6-2 above shows, the tag we search for is geometry and lane section. With the help of XML parser and regex tools, it is simple to locate and access the data of each two tag.

To calculate the complete reference line of each road, it necessary to know how the line is formed in the OpenDRIVE. Every reference line could process few different segments, like straight line or arc. The simplified map defines these segments via four common attributes and three mathematic description: line, poly3 and arc. Four common tags are *s, x, y*, hdg, length. These four elements altogether define the starting position and orientation of a segments.

Table 6-1: Common attributes of lane geometry

Name	Unit	Value	Description
s	m	[0;∞[	*s*-coordinate of start position in Frenet
x	m	]-∞;∞[	Starting position *x* coordinate
y	m	]-∞;∞[	Starting positon *y* coordinate
hdg	rad	]-∞;∞[	Start orientation
length	m	[0;∞[	Length of this road segment

As for line, only four common attributes are enough: when the starting position, pose and length was given, the line was completely defined in 2-D space. Compared to the straight line, the arc segment s requires extra attributes curvature. With starting position and orientation, we could get accurate centroid coordinate. To acquire discrete points on the curve, the angle β is necessary, calculate by arc length times curvature. Based on mentioned two parameters, we could acquire accurate coordinate of any point on the arc.

Table 6-2: Arc and Poly3 extra attributes

Type	Name	Unit	Value	Description
Arc	curvature	1/m	]-∞;∞[	Curvature of the starting point
Poly3	*a*	m	]-∞;∞[	Polynom parameter a
	b	1/m	]-∞;∞[	Polynom parameter b
	c	1/m²	]-∞;∞[	Polynom parameter c
	d	1/m³	]-∞;∞[	Polynom parameter d

Compared to arc and straight line, Poly3 has much edge over presenting complex and irregular points in the real world. Therefore the calculation would be a bit complex. The idea of Poly3 calculation is similar to the Frenet, applying additional coordinate aixes u, v to simplify the calculatoin. The u aixs is point at the direciton of the starting orientation of the curve, while the orgin point is the starting point. As the formula below shown, the $v,\ x, y$ value are all function to u. This is also easy to acquire the normal vector direction of each point on the curve.

$$v(u) = a + bu + cu^2 + du^3 \tag{6-1}$$

$$beta = hdg + \arctan\left(\frac{v(u)}{u}\right) \tag{6-2}$$

$$x(u) = x(u=0) + \sqrt{u^2 + v(u)^2}\sin(beta) \tag{6-3}$$

$$y(u) = y(u=0) + \sqrt{u^2 + v(u)^2}\cos(beta) \tag{6-4}$$

where:

$x(u=0)$ is the starting position of this segment.

$y(u=0)$ is the starting position of this road segment.

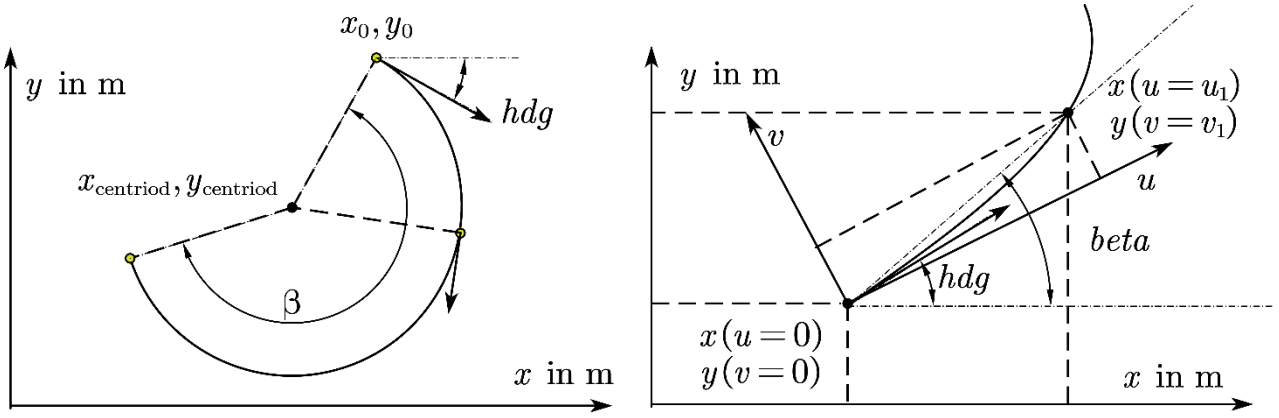


Figure 6-3: Arc (left hand), Poly3(right hand)

Besides the reference line, OpenDRIVE offers convenient way to calculate points' position of other road element, for example the center line of each lane. The center line of each line is parallel to the reference in each road segment. We could acquire the normal vector direction of each point on the curve at first. The multiplication of normal vector and offset, which is the distance between and target point and the projected point on the reference line. For instance, for the first lane on the right hand, the offset is half the lane width, the second lane with one and a half.

When the global path planer generates a route, the map parser could calculate discrete points on every segment on each road. Followed the given road connecting order, all these points would be combined into a two dimensioned array of points in cartesian coordinate. Later the array would be modified into desired data structure and sent to destination client, for example trajectory planner.

6.2 Trajectory Generator

6.2.1 Generation of Lateral Movement in Frenet Coordinate

The lateral trajectory generation is essentially responsible for lane changing and obstacle avoidance. The trajectory planner requires a start lateral state $D_0 = \left[d(t_0), \dot{d}(t_0), \ddot{d}(t_0)\right]$ and an end lateral state $D_1 = \left[d(t_1), \dot{d}(t_1), \ddot{d}(t_1)\right]$ to generate the lateral trajectory, where d denotes the lateral movement.

To ensure the smoothness of the generated trajectory, the trajectory planner chooses quintic polynomial interpolation as the core of the built-in trajectory generation. For the lateral movement, the expression $d(t)$ is:

$$d(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{6-5}$$

meanwhile, the first differential $\dot{d}(t)$ and the second differential $\ddot{d}(t)$ are expressed as:

$$\dot{d}(t) = a_1 + a_2 t + a_3 t^2 + a_4 t^3 + a_5 t^4 \tag{6-6}$$

$$\ddot{d}(t) = a_2 + a_3 t + a_4 t^2 + a_5 t^3 \tag{6-7}$$

where a_i denotes the parameters of the quintic polynomial.

The calculation processes $d(t), \dot{d}(t), \ddot{d}(t)$ are written in a matrix form, which are presented as follows.

$$\begin{aligned}\begin{bmatrix} d(t) \\ \dot{d}(t) \\ \ddot{d}(t) \end{bmatrix} &= \begin{bmatrix} 1 & t & t^2 \\ 0 & 1 & t \\ 0 & 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} + \begin{bmatrix} t^3 & t^4 & t^5 \\ 3t^2 & 4t^3 & 5t^4 \\ 6t & 12t^2 & 20t^3 \end{bmatrix} \cdot \begin{bmatrix} a_3 \\ a_4 \\ a_5 \end{bmatrix} \\ &= \boldsymbol{M}_{d1}(t) \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} + \boldsymbol{M}_{d2}(t) \cdot \begin{bmatrix} a_3 \\ a_4 \\ a_5 \end{bmatrix}\end{aligned} \tag{6-8}$$

To simplify the calculation, usually, let the time parameters t_0, t_1 and the first three parameters of quintic polynomial a_0, a_1, a_2 be set as the follows.

$$t_0 = 0, \qquad t_1 = \tau, \qquad a_0 = 0, \qquad a_1 = \dot{d}(0), \qquad a_2 = \frac{\ddot{d}(0)}{2} \tag{6-9}$$

The rest parameters of quintic polynomial a_3, a_4, a_5 can be solved with:

$$\begin{bmatrix} a_3 \\ a_4 \\ a_5 \end{bmatrix} = \boldsymbol{M}_{d2}^{-1}(\tau) \cdot \left[\begin{bmatrix} d(\tau) \\ \dot{d}(\tau) \\ \ddot{d}(\tau) \end{bmatrix} - \boldsymbol{M}_{d1}(\tau) \cdot \begin{bmatrix} d(0) \\ \dot{d}(0) \\ \ddot{d}(0) \end{bmatrix} \right] \tag{6-10}$$

If the start state and the vehicle's end state are given, a quintic-polynomial can always be generated to connect the start state and the end state. Figure 6-4 exhibits an example of generated lateral trajectory in MATLAB.

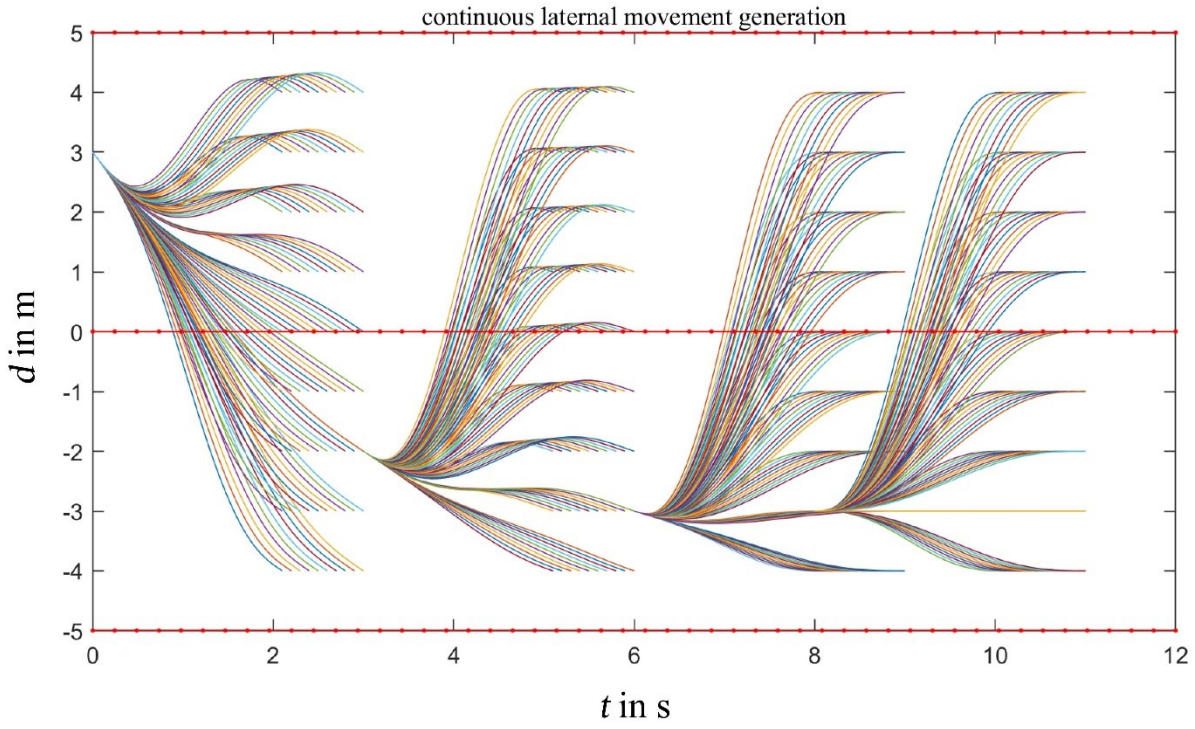


Figure 6-4: Continuous lateral movement generation

6.2.2 Generation of Longitudinal Movement in Frenet Coordinate

Longitudinal movement in the Frenet coordinate is different from those lateral movement. The longitudinal direction is responsible for distance keeping, merging, and stopping at a certain position.

A start longitudinal state $S_0 = [s(t_0), \dot{s}(t_0), \ddot{s}(t_0)]$ and an end longitudinal state $S_1 = [s(t_1), \dot{s}(t_1), \ddot{s}(t_1)]$ are the requisites of the trajectory planner, where s denotes the arc length of the trajectory, which is also the vertical axis of the Frenet coordinate. According to the following formula, which varies the end constraints by different Δs_i and T_j, a set of longitudinal trajectories can be generated analogously to the lateral trajectories.

$$[s(t_0), \dot{s}(t_0), \ddot{s}(t_0), T]_{ij} = \left[\left[s(T_j) + \Delta s_i \right], \dot{s}(T_j), \ddot{s}(T_j), T_j \right] \tag{6-11}$$

$$\begin{bmatrix} a_3 \\ a_4 \\ a_5 \end{bmatrix} = \boldsymbol{M}_{s2}^{-1}(\tau) \cdot \left[\begin{bmatrix} s(\tau) \\ \dot{s}(\tau) \\ \ddot{s}(\tau) \end{bmatrix} - \boldsymbol{M}_{s1}(\tau) \begin{bmatrix} s(0) \\ \dot{s}(0) \\ \ddot{s}(0) \end{bmatrix} \right] \tag{6-12}$$

where Δs_i denotes the offset of the final position and T_j indicates the time interval between the start state and the end state.

Figure 6-5 presents an example of longitudinal movement in MATLAB.

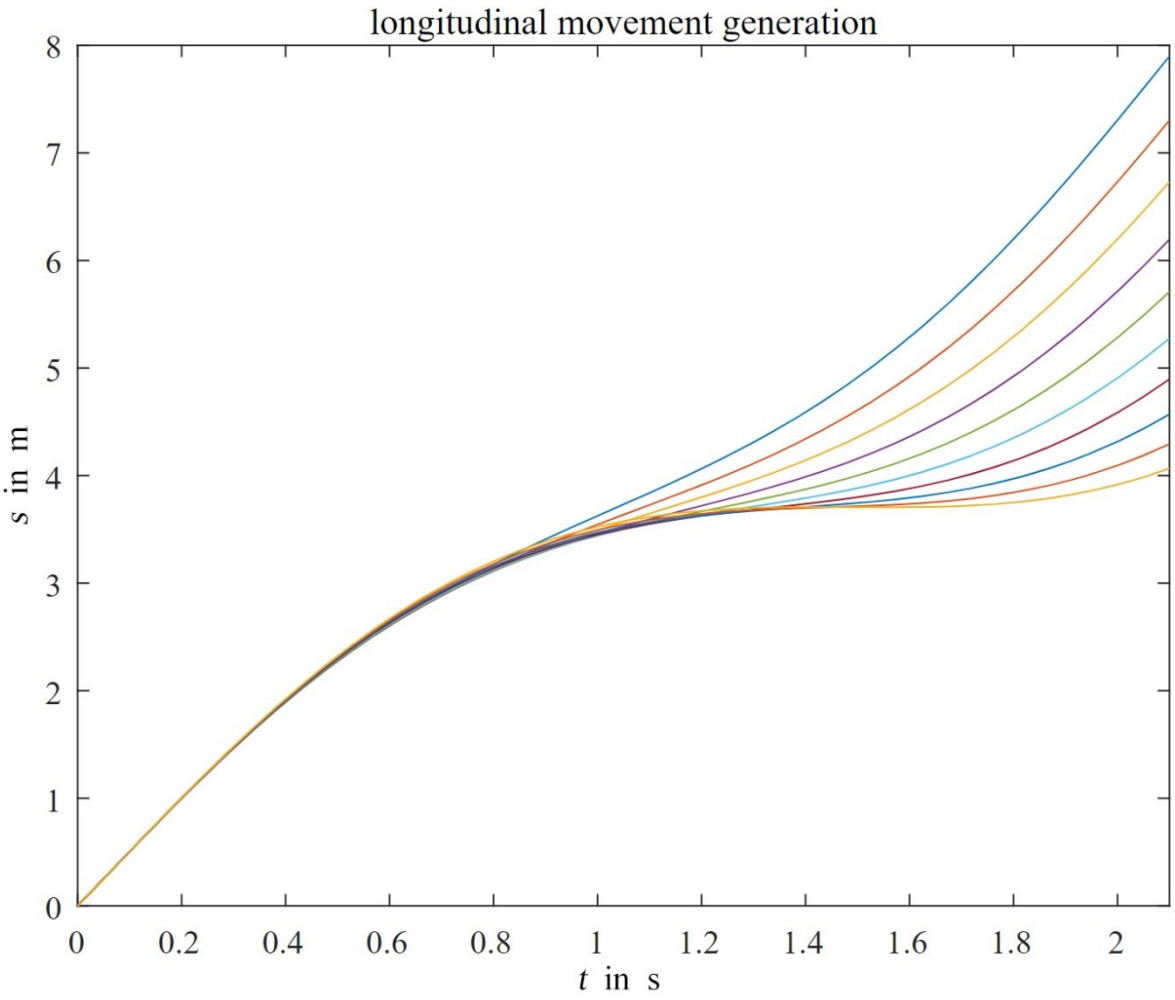


Figure 6-5: Continuous longitudinal movement generation

6.2.3 Combining the Lateral and Longitudinal Trajectories

Figure 6-6 and Figure 6-7 show that when a start state and different end states with a specific time interval are given, multiple trajectories will be generated in lateral and longitudinal direction, so the next step is to combine the lateral and the longitudinal trajectories in cartesian coordinate. After transforming the trajectories from Frenet coordinate to cartesian coordinate, lateral and longitudinal trajectories with the same start state, end state and time interval will be combined through cubic spline,

which are later exported for security check and ranked by cost function. Fig.5 and Fig.6 show the combination of lateral movement and longitudinal movement as an example.

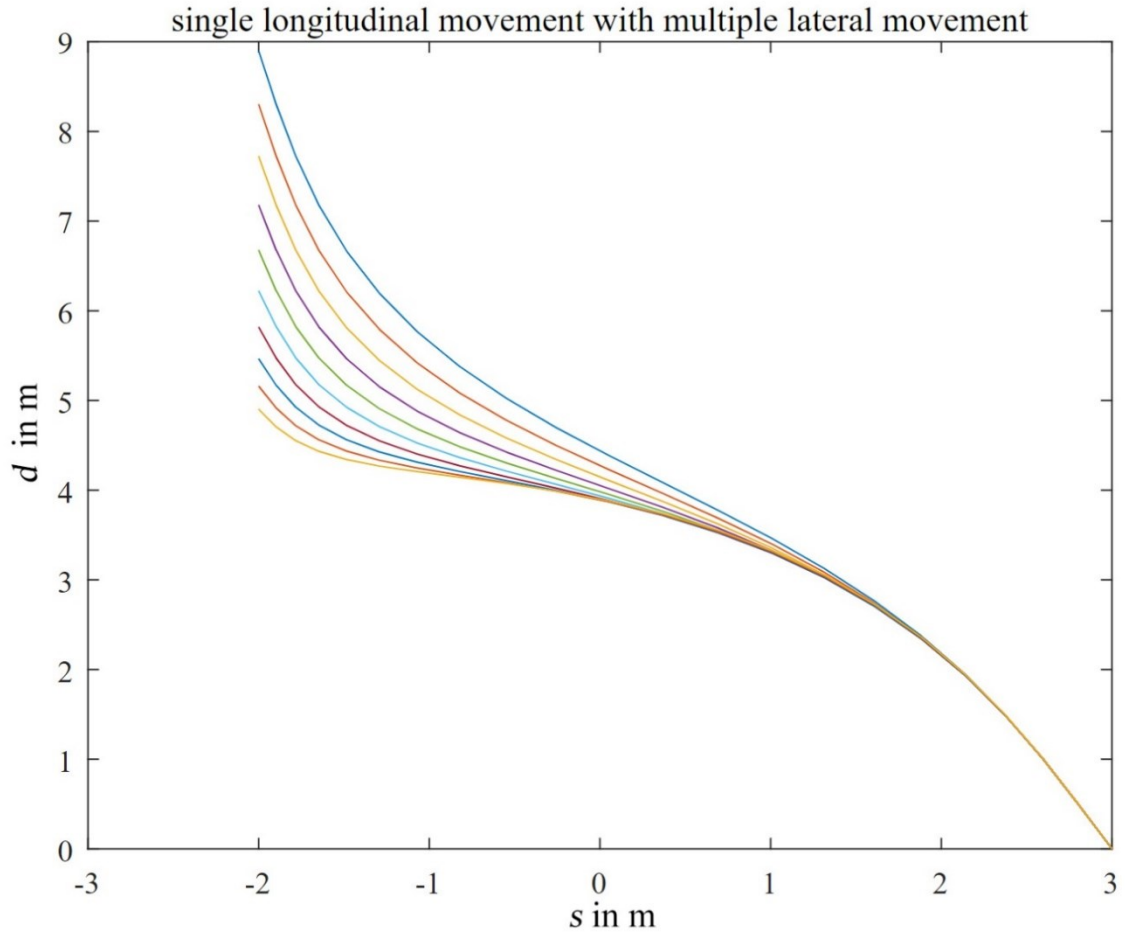


Figure 6-6: Single longitudinal movement with multiple lateral movement

The above illustration shows that different quintic-polynomials are generated to connect the start state and the end state when different offset d (different start state S_0) with the same end state S_1 are chosen in Frenet coordinate.

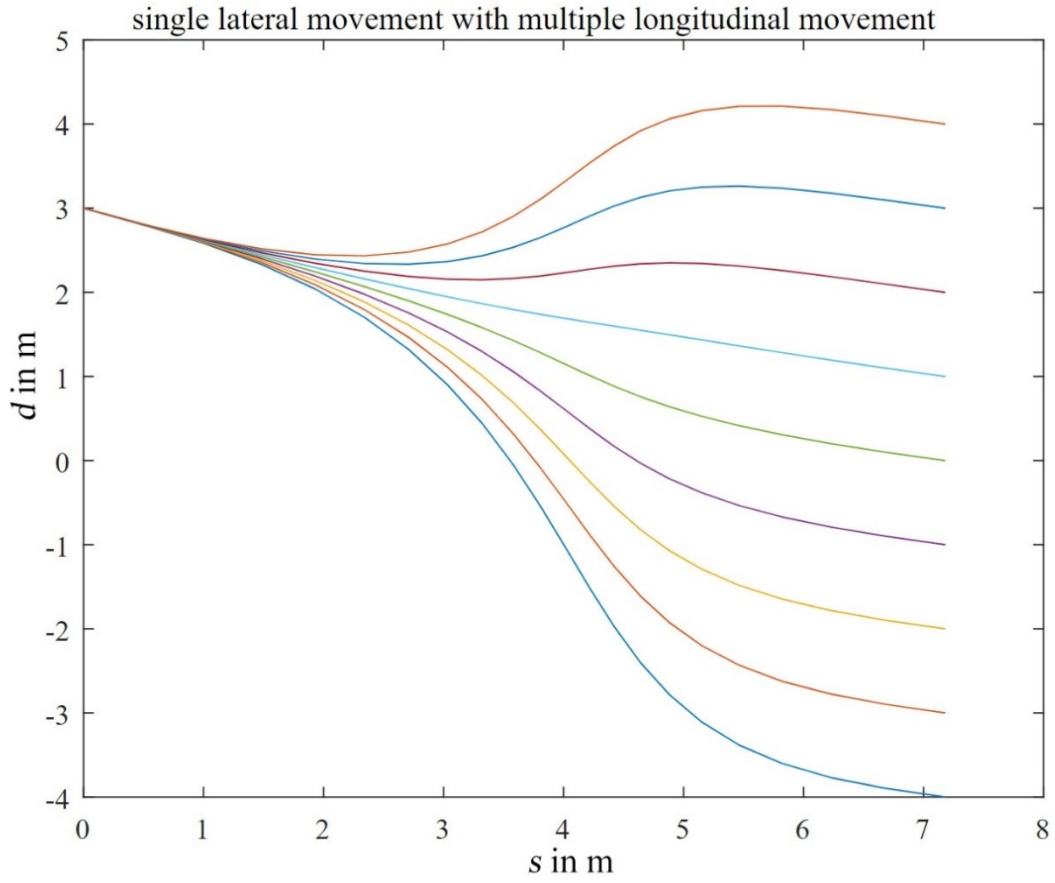


Figure 6-7: Single lateral movement with multiple longitudinal movement

Figure 6-7 reveals that: when the same offset d (same start state S_0) with different end states S_1 are chosen in Frenet coordinate, different quintic-polynomials are generated to connect the start state and the end state. From those two figures, it is clear to determine how the start state and the end state influence the trajectory generation. Moreover, it is reminded once again that the aforementioned optimal control approach's focus is not to minimize or maximize the cost function, which achieves the optimal result through convex optimization iteration; however, it generates the possible trajectories randomly and ranks the obtained results by comparing the cost to find out the best one.

6.3 Cost Function Design

The design of the cost function is an essential step in the process of generating the trajectory. Some guidelines should be considered in this process, such as driving comfort, total time cost, and the lateral displacement from actual trajectory to centerline. Mathematically conversing, the best way to describe the driving comfort is the jerk, characterized by the acceleration's alteration over time. Furthermore, since the problem was adapted from Cartesian coordinate to Frenet coordinate, the jerk can only be analyzed in Frenet coordinate as $\dddot{s}$ and $\dddot{d}$. During the time interval $t_1 - t_0$, the integral of the jerk J_t is calculated as:

$$J_t = \int_{t0}^{t1} \dddot{p}^2 \, dt, \tag{6-13}$$

where $\dddot{p}$ means the square of tangential acceleration derivative $\dddot{s}$ or the square of normal acceleration derivative $\dddot{d}$.

From the above viewpoint, the cost function for lateral movement is expressed as follows.

$$C_d = k_j J_t + k_t T + k_d d^2 \tag{6-14}$$

where C_d denotes cost for lateral movement, T signifies the time interval, d^2 indicates the lateral trajectory's offset; moreover, k_j, k_t, and k_d express weight parameters in the lateral cost function above.

Analogous to the lateral cost function, the longitudinal cost function has the same mathematical form.

$$C_s = k_j J_t + k_t T + k_s S \tag{6-15}$$

where C_s describes the cost in longitudinal direction, S represents the difference between the start position $s(t_0)$ and the end position $s(t_1)$; furthermore, k_j, k_t, and k_s determines the weight parameters.

Ultimately, the total cost function for the combinative trajectory is defined as follows:

$$C = k_{\text{lateral}} C_d + k_{\text{longitudinal}} C_s \tag{6-16}$$

where k_{lateral} and $k_{\text{longitudinal}}$ indicate the weight of the lateral cost function and the importance of longitudinal cost function.

For the selection of the weights of the cost function, we simply use the value given by GitHub code, where $k_j = k_t = 0.1, k_s = k_d = k_{\text{lateral}} = k_{\text{longitudinal}} = 1$, it is very objective and in our python demo the model works well. For the further work of the selection of these parameters, more search and experiments should be conducted.

6.4 Legality and Security Check

For the security check, the trajectory planner must ensure that the generated trajectory has no intersection with the objects around the vehicle. Instead of adding heuristic penalty terms to the cost function in the vicinity of other obstacles, which may lead to complex parameter adjustments as well as unpredictable behavior[20], it is better to set a safety distance to control the distance between the vehicle and the surrounding objects. If there exists one point on the trajectory whose distance to the obstacles is smaller than the safety distance, then the whole trajectory will be considered unsafe and removed. Additionally, the trajectory should also satisfy the limitations of the vehicle dynamics, which means the curvature of the trajectory, acceleration and velocity of the vehicle must meet the actual situation,

[20] M. Werling et al.: Optimal Trajectory Generation for Dynamic Street Scenarios in a Frenét Frame (2010).

a set of upper bounds are set to ensure that the trajectories are executable. The illustration exhibits this process as follows.

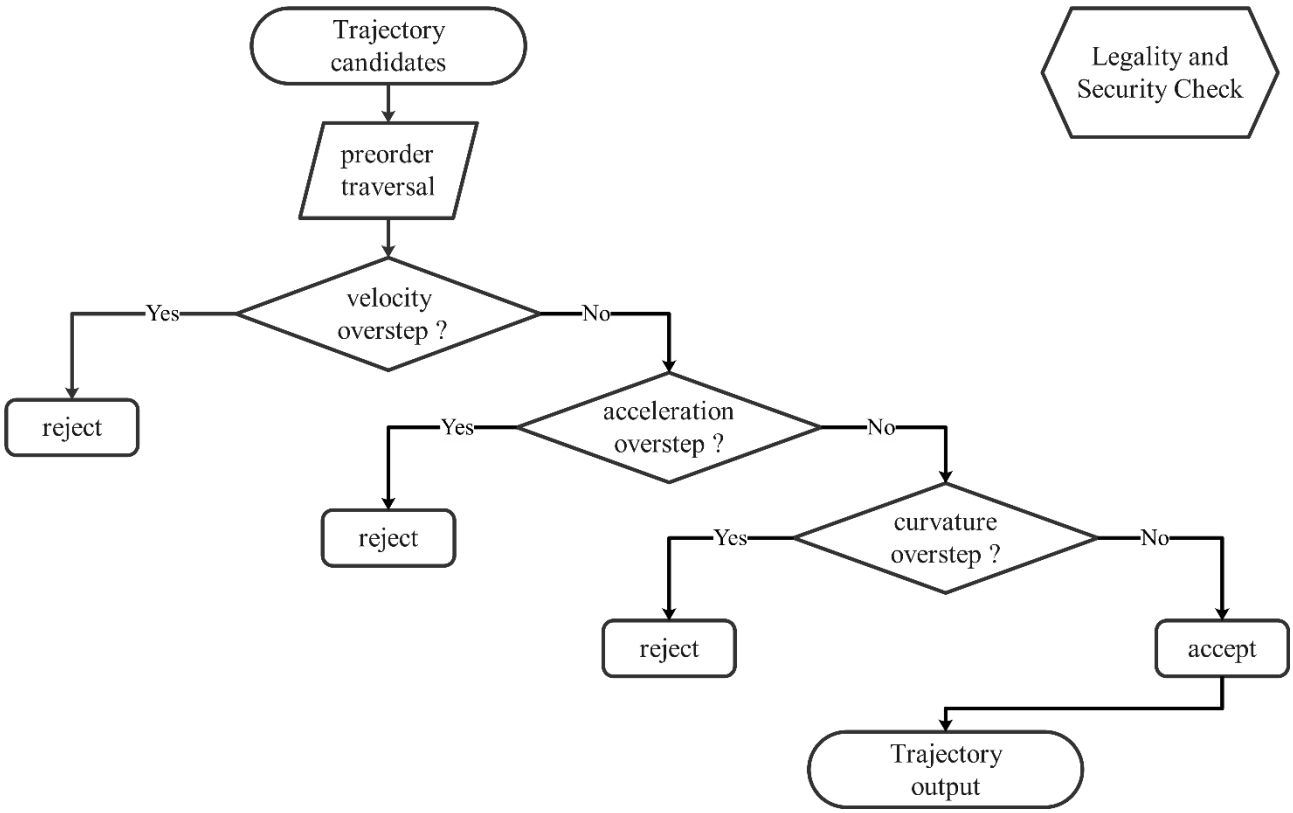


Figure 6-8: Trajectory check

7 Algorithm Validation

7.1 Implementation Environment

In order to test the final performance of our finite state machine and the feasibility of generating the corresponding trajectories, CarMaker and ROS are selected as the simulation test drive methods.

This software is mainly used for the development of vehicles and systems in the automobile industry. In addition to a development environment and corresponding models for replicating a real driving test in the virtual world, CarMaker includes various tools for parameterization, simulation, analysis, and diagnosis of all model components. This software application can be divided into driver assistance systems, automatic driving functions, power systems, driving dynamics, and other fields.

7.1.1 The Virtual Vehicle Environment – VVE

A virtual vehicle is an actual vehicle's computer model, and its behavior matches with its real-world counterpart. With CarMaker, the virtual vehicle comprises mathematical models that contain the equations of motion, kinematics, and other mathematical formulas that define the multibody system. Then, the model is parameterized with data directly related to the vehicle to be researched. In this way, CarMaker can test any vehicle with verified parameter sets and easily switch between virtual vehicles by changing the vehicle model's parameter data.

In addition to the vehicle model, CarMaker could digitally and accurately build real-world test scenarios, including the whole surrounding environment, such as road, traffic, and maneuver. The simulation of VVE is shown in Fig. 7-1.

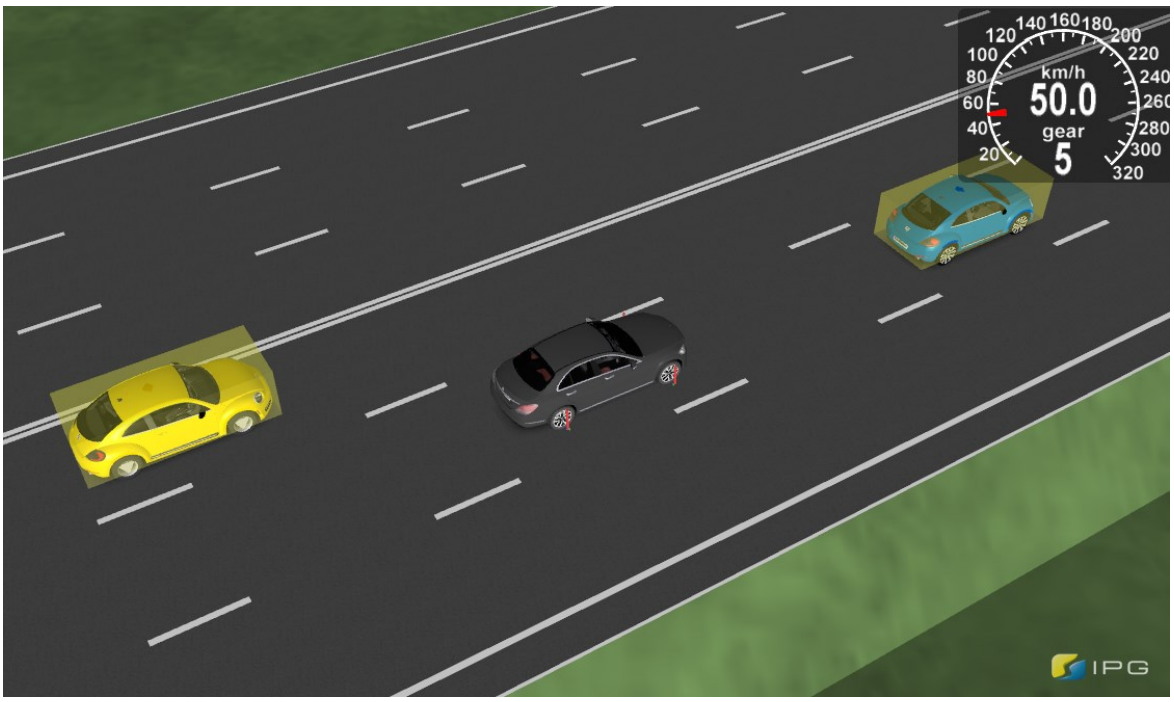


Figure 7-1: The simulation of VVE

7.1.2 The CarMaker Interface Toolbox- CIT

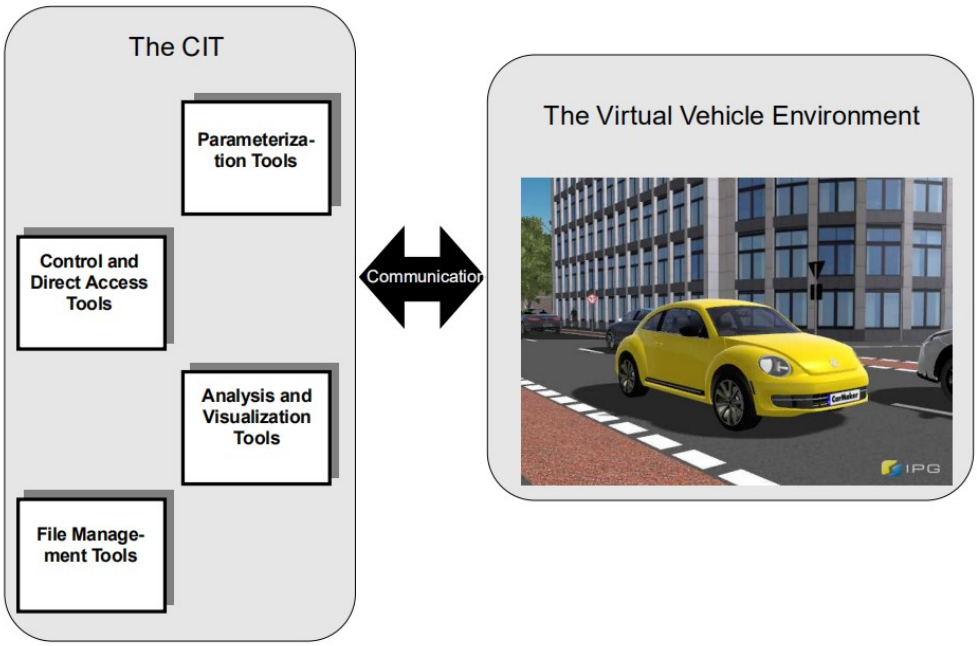


Figure 7-2: The CIT and VVE

This part includes all the tools that are used to manage the VVE. Figure 7-2 shows the CIT in relation to the VVE. What we use the most in our test is represented as the following:

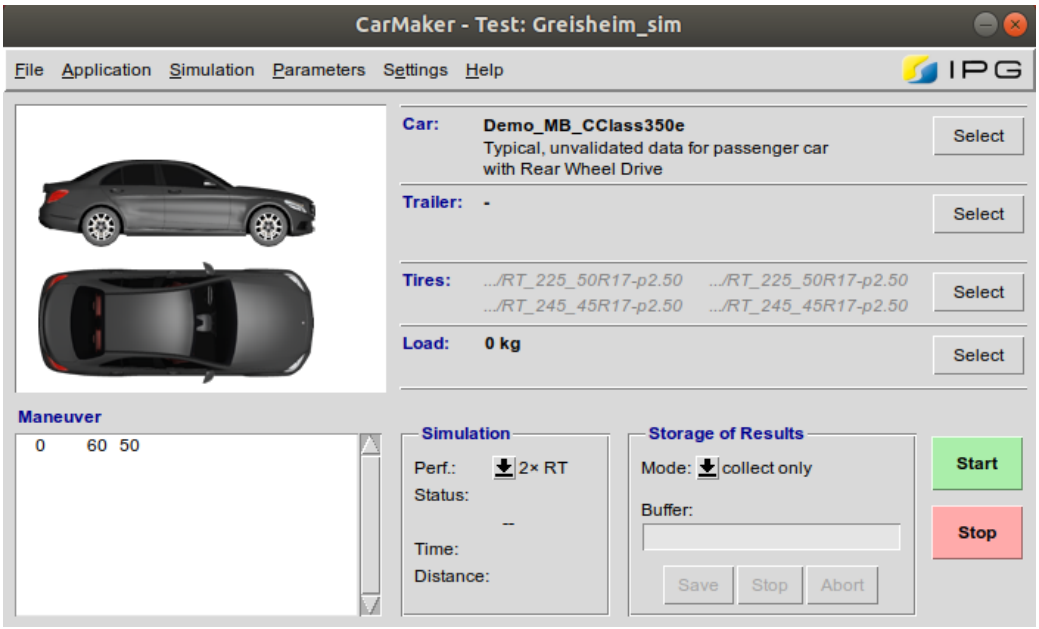


Figure 7-3: CarMaker GUI

- **The CarMaker GUI:** The Figure 7-3 shows the main graphical user interface which is used to control the actions and positions of VVE and other traffic, such as select the parameter data of virtual vehicles, define or select the virtual road, set the virtual driver parameters, define or load maneuver actions.

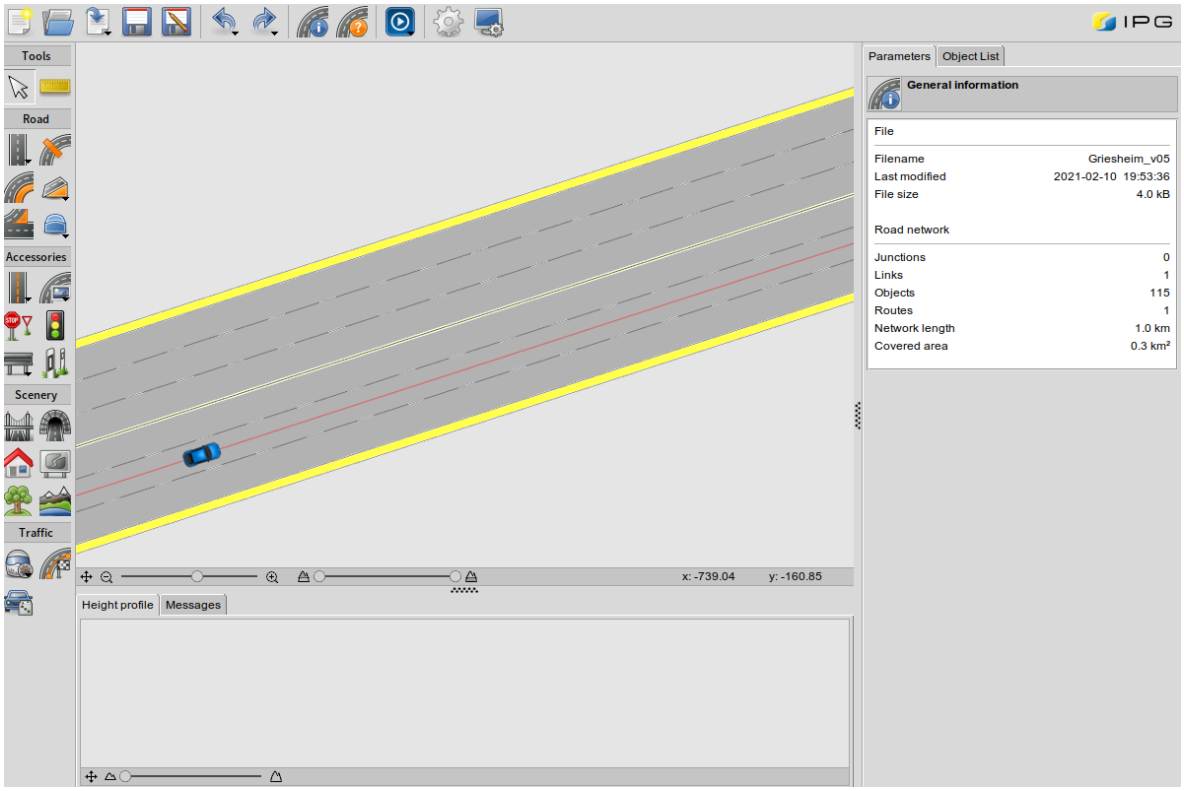


Figure 7-4: Scenario editor

- **Scenario editor:** The accurate simulation environment is built by loading the existing map data or established manually, and the information about the road and traffic lights could be added as will.

- **IPGMovie:** Real-time 3D animation of the VVE, which is shown as Figure 7-1.

▪ ROS publisher and ubscriber

ROS is an open-source meta-operating system of robot. It provides services from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. The role that ROS plays in

this project is the bridge that connect different modules, such as the communication between CarMaker and FSM, FSM and trajectory planner, trajectory planner and CarMaker.

To further introduce the working mechanism of the project, the definition of ROS node and ROS topic should be firstly clarified.

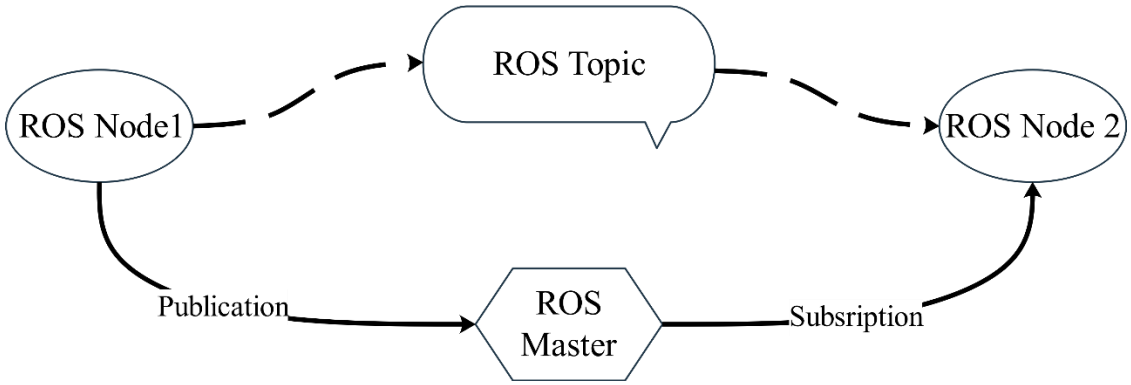


Figure 7-5: ROS nodes and Messages

ROS starts with the ROS master. Host allows all other ROS software (nodes) to find and communicate with each other. A node is a process that performs computation and nodes are combined together into a graph and communicate with one another using streaming topics. Here, publisher("talker") node is created to continuously broadcast messages while subscriber("listener") receives messages.

In the validation, the FSM node `/straight_fsm_node` *and* `/cross_fsm_node` both are subscriber and publisher nodes, which receive the messages about ego car and the surrounding vehicles from CarMaker as well as the map information from `/waypointer`. Besides, they also publish the corresponding messages to the `/trajectoryPlanner` node, which include the target position, velocity and the next update state of the ego car. A schematic, modalized drawing of the working environment of the project in ROS is shown in Figure 7-6 below:

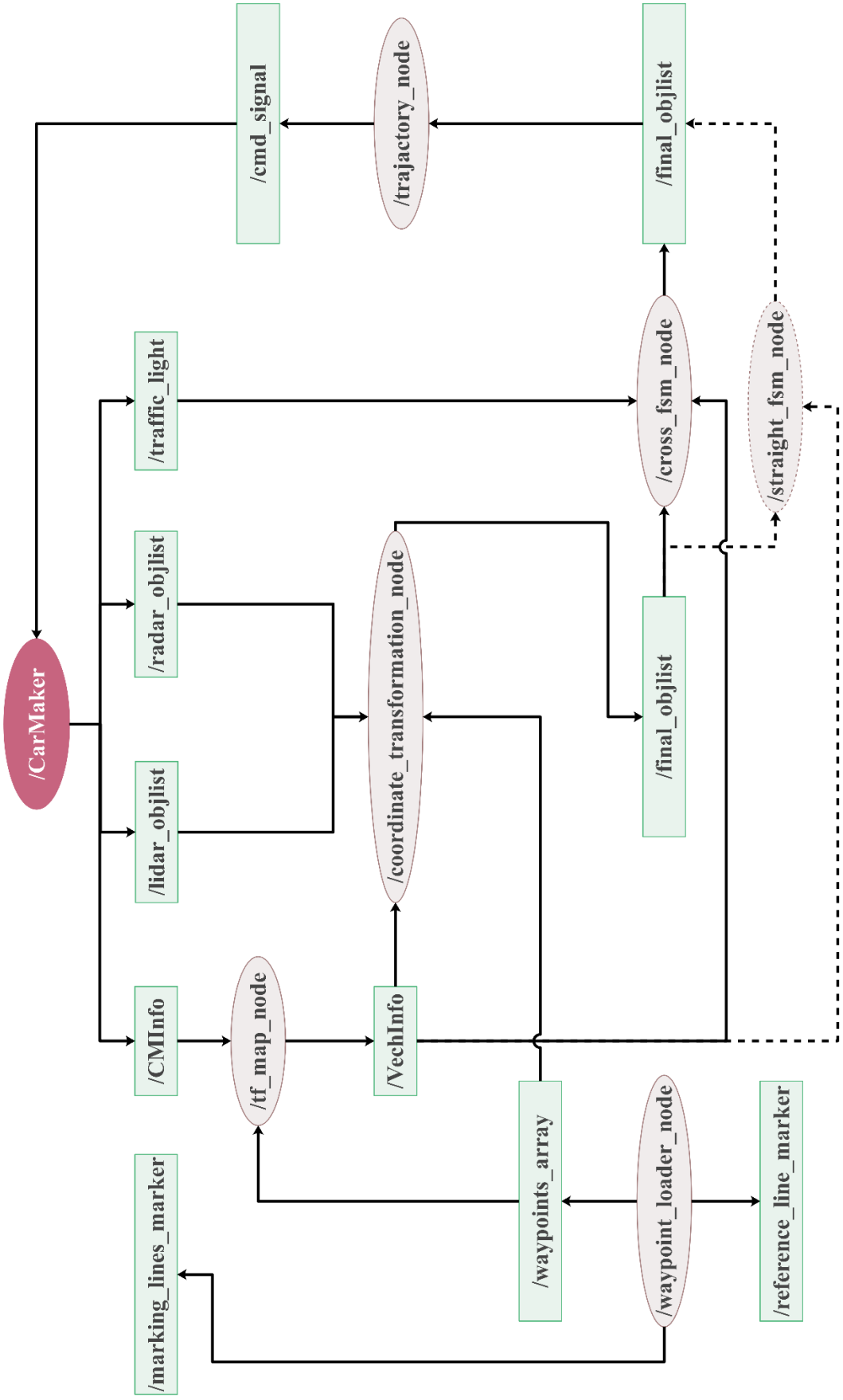


Figure 7-6: Modularized working environment of the project in ROS

7.2 Scenario Edit in CarMaker

To implement simulations, the Griesheim map is imported into CarMaker. This is a single carriage-way with two lanes and the 3 forks with 3 ways, which makes the road a big loop. On each intersection there are some traffic lights. The followings are the road parameters:

Table 7-1: Parameters of Links

Links	length(m)	Start angle (deg)
Link 0	146.91	197.92
Link 1	20.73	108.07
Link 2	44.72	218.57
Link 3	878.18	197.80
Link 4	732.48	218.25
Link 5	146.04	132.66

Besides, there are in total 7 traffic light in the map. Each traffic light has a green phase duration of 25 seconds, a yellow phase duration of 3 seconds, a red phase duration of 25 seconds and a red plus yellow phase of 3 seconds.

The origin of the ego car is located in Link 0 at the start point, as the red point in Figure below shows. The route planed is from start point along link 3, link 4 and link5 and then return the start point.

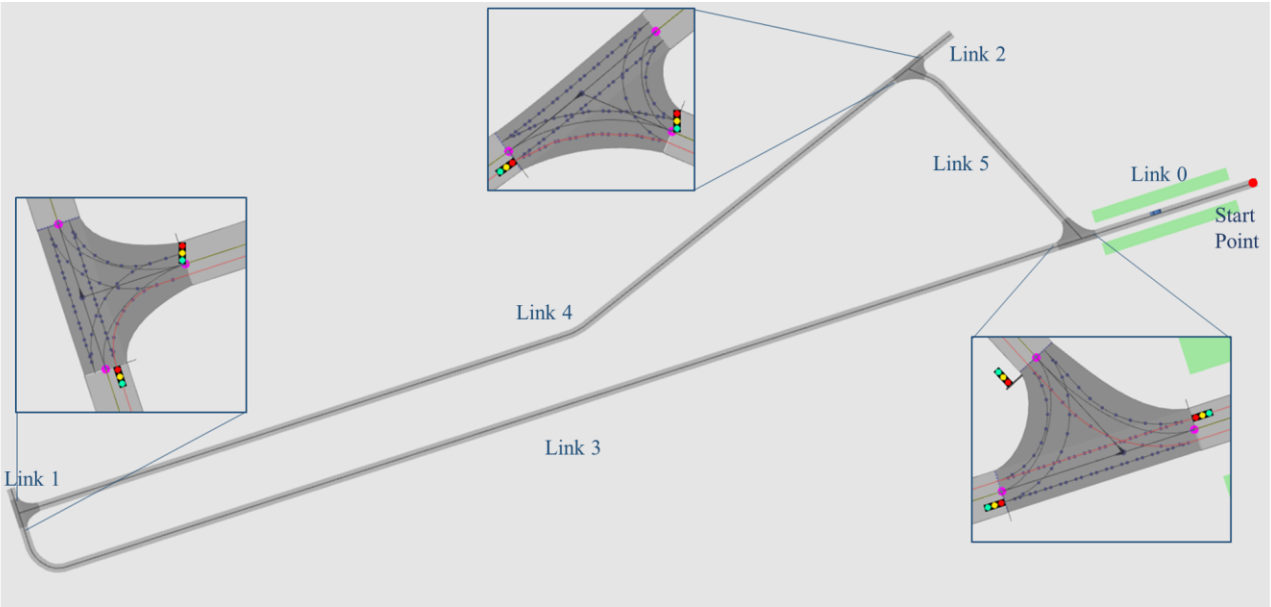


Figure 7-7: Map of Greisheim

7.3 Validation in Straight Scenario

7.3.1 Purpose

The main purpose of this experiment is to investigate the driver's decision-making in a straight-line scene. We chose the overtake situation as our major test scenario while it is a complex situation that could verify prep change lane, change lane and collision check multiple procedures.

7.3.2 Method and Procedure

Through building three lane road and traffic flow around the ego car, the scenario for overtaking is established.

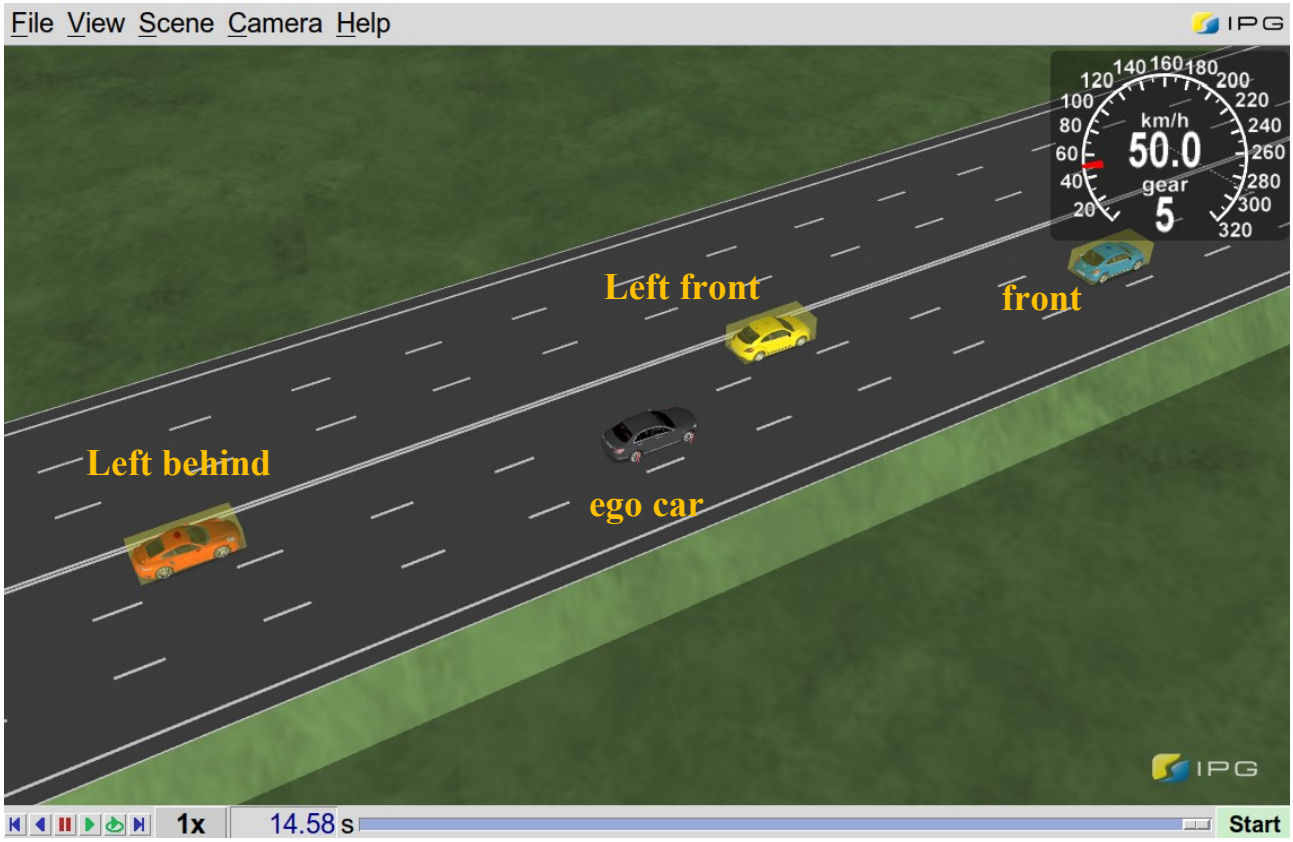


Figure 7-8: The ego car and the surrounding vehicles

- road scenario: The longitudinal length of road is 1000 m, and the distance between the centerline of adjoining lanes is 3.5m.
- traffic: Three other vehicles are added here, which refer to the front obstacle, left front obstacle and left behind obstacle compared to the ego car, which is shown in Figure 7-8.

The initial position of the front vehicle is 200m while our ego car is 0 m. In order to meet the condition of overtaking, the velocity of front vehicle is set to less than 50% of the ego car.

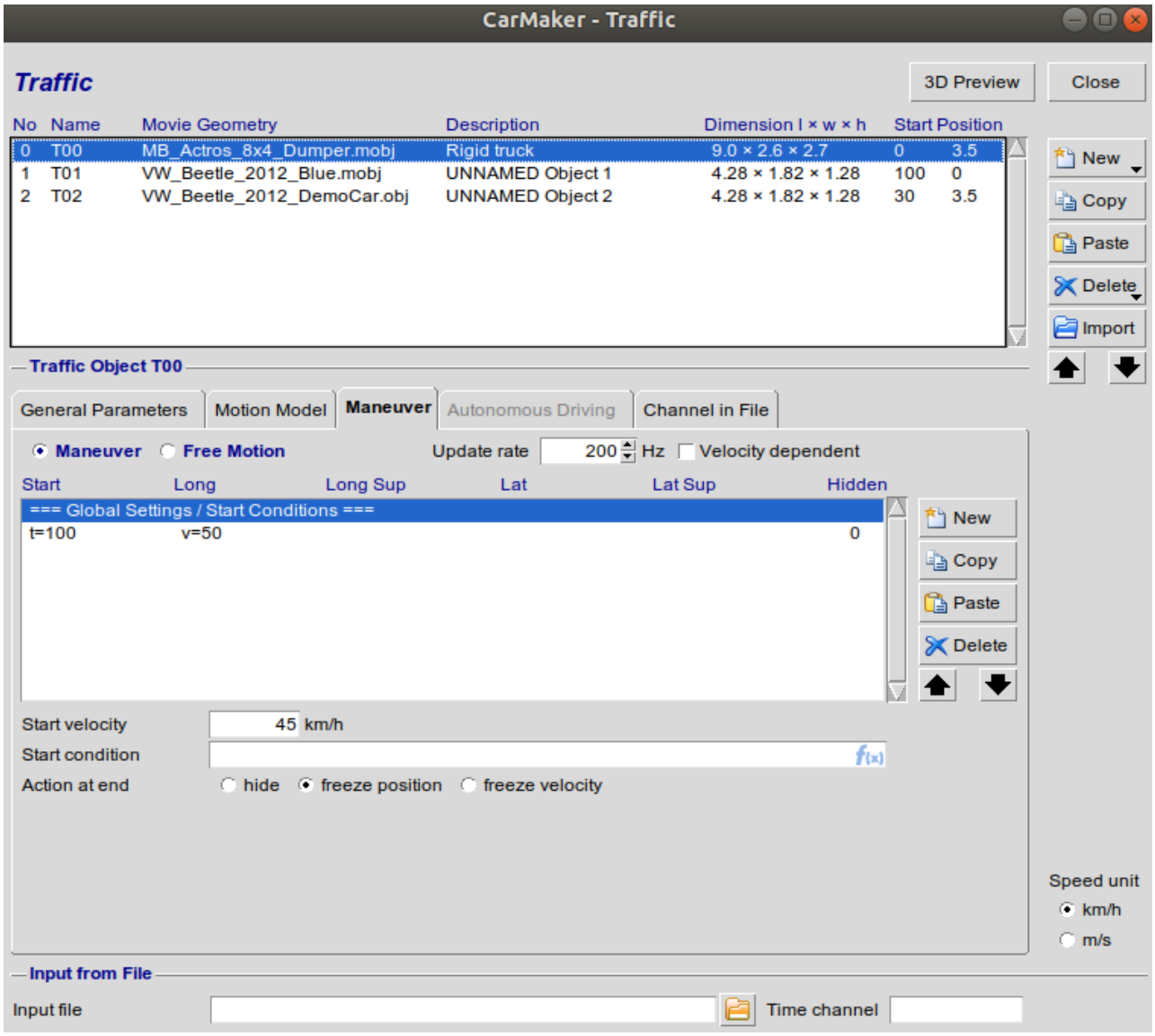


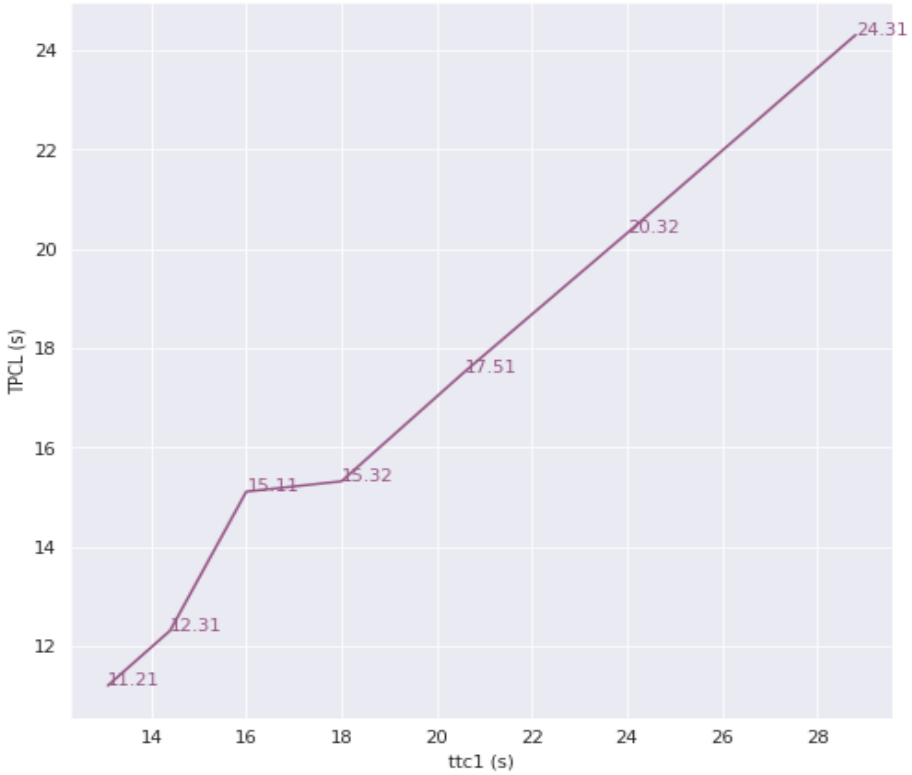
Figure 7-9: The General parameters of traffic-flow

As the calculation of two "gaps" described in chapter 5.2, enough space between other two objects in the left lane are also remain to test whether the self-vehicle could successfully transfer from prep change lane left state to the change lane left state.

Overtaking judgment: When there is a relatively slow obstacle in the front of the ego car, sufficient time to overtake plays an import role here. In order to investigate whether and when the ego car could transfer from keep lane state to prep change lane left state, we have set multiple trials on the different sets of the speed and position of the ego car and the front obstacle.

The speed of the front obstacle is always 5.55m/s (20 km/h) and the initial position is 200 m compared to the ego car. The speed of the ego car was varied between 12.5 and 25 m/s (45-90 km/h). Thus, the time to the front obstacle (TTC1) could be in range from 10.29 to 28.80 s.

In the trial, the time to prep change lane left (TPCL) is recorded. And the relationship between the TPCL and TTC1 is shown in the figure 7-10. As the result of trial in figure 7-9 presents, when ttc1 is in the range of 11.205 and 24.307 s, the ego car could enter in the prep change lane left state and ttc1



is in proportional to TPCL as well. Otherwise, the ego car would stay in the keep lane state or change to the stop state.

Figure 7-10: The line chart of TTC1 and TPCL

Overtaking maneuver: Although the ego car could successfully enter in the PCL state, if the lane change left condition is not met, the overtaking command would still not be implemented. Whether the ego car could change the lane left depends on the situation of the other vehicles in the left lane. Here, the time to the left obstacle (TTC 2) is calculated. The explanation of TTC 1 and TTC 2 is represented in the Figure 7-11. For this calculation, we set both vehicles to the speed of 13.89 m/s (50 km/h) but with different initial distances. Therefore, the time for the ego vehicle to reach the obstacle in front of the left lane along the longitudinal direction varies.

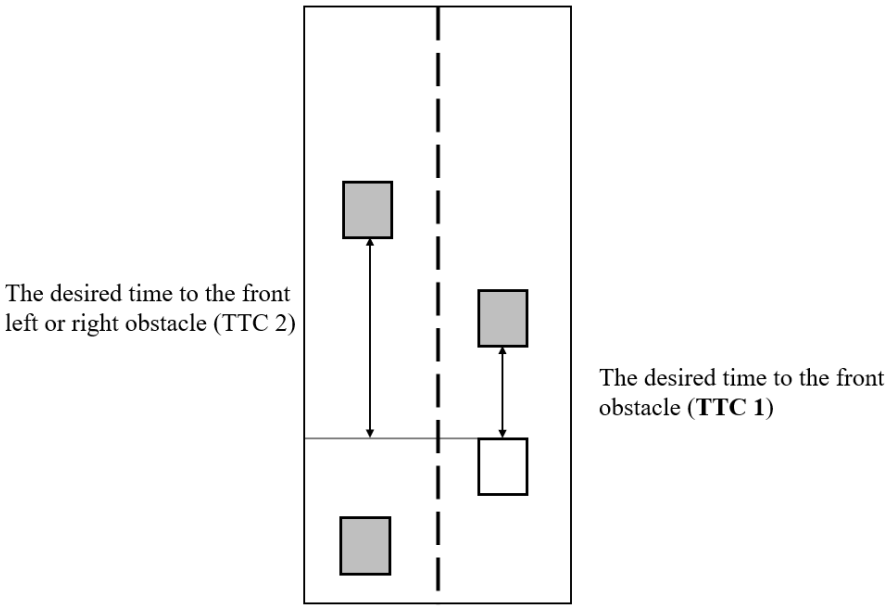


Figure 7-10: The explanation of TTC 1 and TTC2

What stands out in the table 7-2 is that the ego car with different TTC 1 could fail to change the lane left in a certain range of TTC 2.

Table 7-2: TTC 2 and TTC 1

TTC	TTC 2 (s)	PCL->CL
17.5	22.32-27.36	Failed
15.41	19.8-21.6	Failed
13.71	18.48-19.92	Failed

7.4 Validation in Intersection Scenario

This section provides insights into the verification process of the urban autonomous driving algorithm in an intersection scenario. As mentioned in chapter 2, the simulated result in a virtual environment is needed to validate; therefore, the corresponding recorded data is primarily required to compare the behaviors of the automation to the build-in algorithm of CarMaker in critical situations.

7.4.1 Open Loop Validation

As discussed in chapter 4, the urban autonomous driving algorithm has a multi-layer structure, based on structured top-down communication system. In the decision-making process, the instructions of the upper-layer module directly determine the output of the lower module, which means that the correctness of the results of the behavior planner will directly influence the success of the autonomous driving results. Therefore, in the verification phase, the first task is to verify the correctness of the behavior planner.

In order to independently analyze the behavior planner, the verification will be processed in an open-loop mode, which can avoid interference caused by other factors, such as trajectory planner. The open-loop method means that the simulator runs autonomously under the built-in program. Simultaneously, the behavior planner only analyzes the recorded relevant information, but the result, such as state and target, will not be input into the simulator.

There are two main aspects to consider during validation of behavior planner:

- Correctness of state transition
- Correctness of predicted motion target

7.4.1.1 Correctness of State Transition

The correctness of state transition refers to the match between the state entered by the FSM and the dynamic of the ego car with its surrounding environment. According to statistical principle, there are two quantitative critical indicators named “false-positive rate” (FPR) and “false-negative rate” (FNR) are built to evaluate the correctness of state transition.

The FPR means that no object in the reality but an object wrongly exists in the decision of FSM. In a virtual simulation environment with open-loop control, the vehicle with a built-in algorithm has gone ahead, but FSM remains in the stop state.

The FNR indicate that object exists in the reality, but FSM fails to perceive it. In other words, the vehicle in CarMaker brakes but FSM enters in other states.

Since CarMaker cannot directly output the vehicle brake signal in the ROS environment, the inverse Time-to-Collision (iTTC) was chosen as a validation reference. When iTTC is significantly reduced, the vehicle in front is escaping from the ego car. Conversely, if iTTC increases considerably in a short

period, the vehicle ahead has braking action. Meanwhile, the ego car should also make a braking response. The graph below distinctly exhibits the development of iTTC and the state from behavior planner.

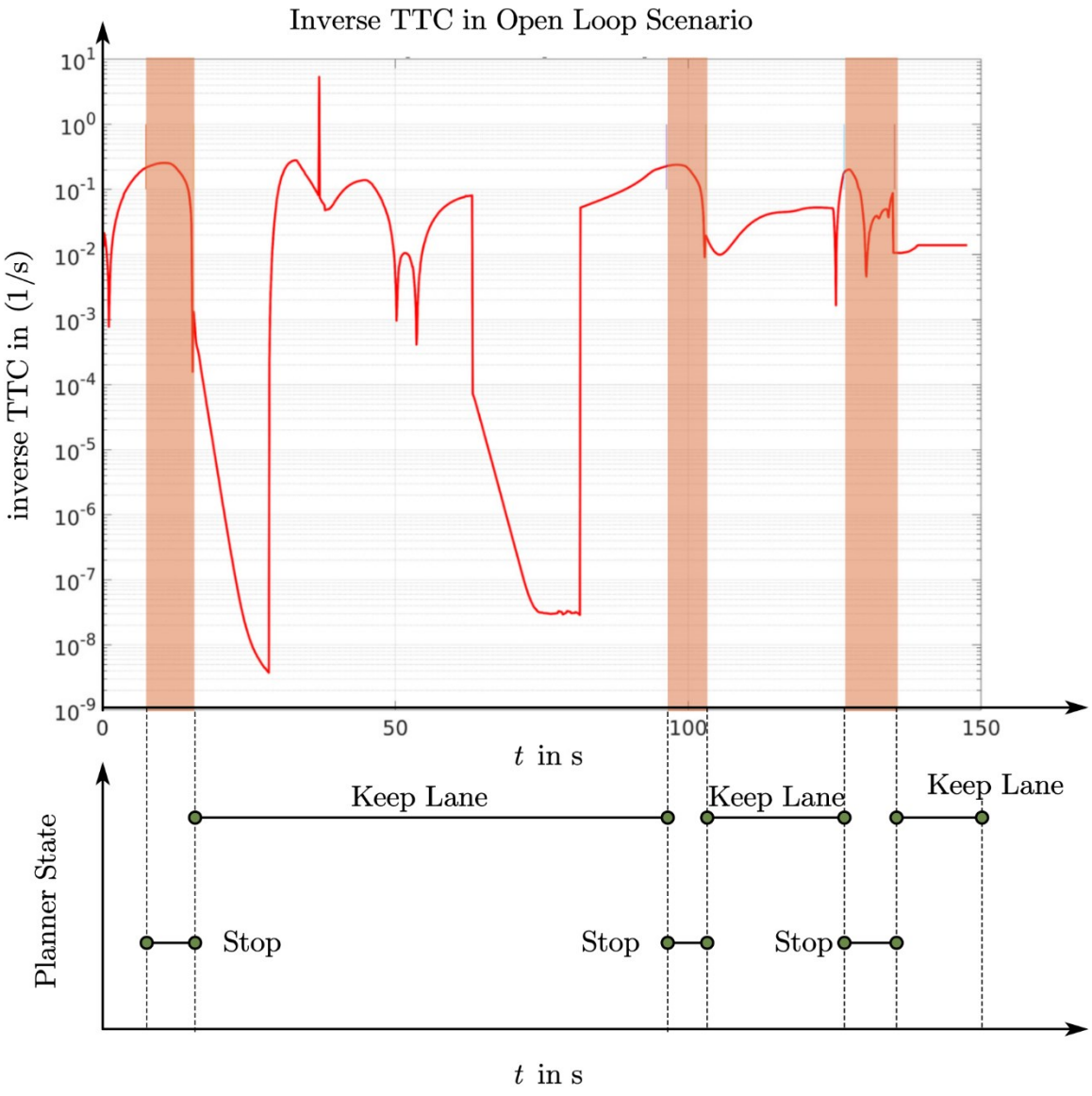


Figure 7-10: inverse TTC vs. stop state

As shown from the above figure, the state will progress to the stop state when TTCR is close to 0.2 and exit the stop state when it is less than 0.1. Here chose 0.2 as the threshold for entering the stop state, and 0.1 as the threshold for leaving the stop state. Based on these thresholds, the stop state and non-stop state values are counted. Conclusively, the following content can be obtained.

Table 7-2: Correctness of state transition

	Behavior Planner	
	Keep Lane	Stop
CarMaker Go Ahead	1232	57
CarMaker Stops	42	145
Result	3.2967 %	28.2178 %
Description	FNR	FPR

As noted from the above table, the FNR of the behavior planner is about 3%, which means that the FSM instructions only have a small difference from CarMaker built-in algorithm in case of open loop. The animation of the IPG Movie proves that those difference only appears in the stop phase: When the Ego car is in a stopped state, CarMaker will always give instructions to stop at the same place unless the leader vehicle has already started. However, the behavior planner will provide a keep lane instruction to obtain a short stop gap even the ego car has stopped. The higher FPR in the table represents another situation: the FSM has a more conservative condition for vehicles. By comparing the status of IPG Movie, FSM enters the stop state earlier than CarMaker in the driving state, which means a safer and more conservative state decision. Therefore, the behavior planner has a higher traffic efficiency than CarMaker built-in algorithm without any compromising of security.

In addition to security, from the perspective of qualitative analysis, the legality of instructions must be considered within the correctness of the state transition. The current related state, combined with the vehicle velocity and the signal light, is sufficient to evaluate whether the behavior planner's instruction complies with traffic laws and regulations, illustrated as following.

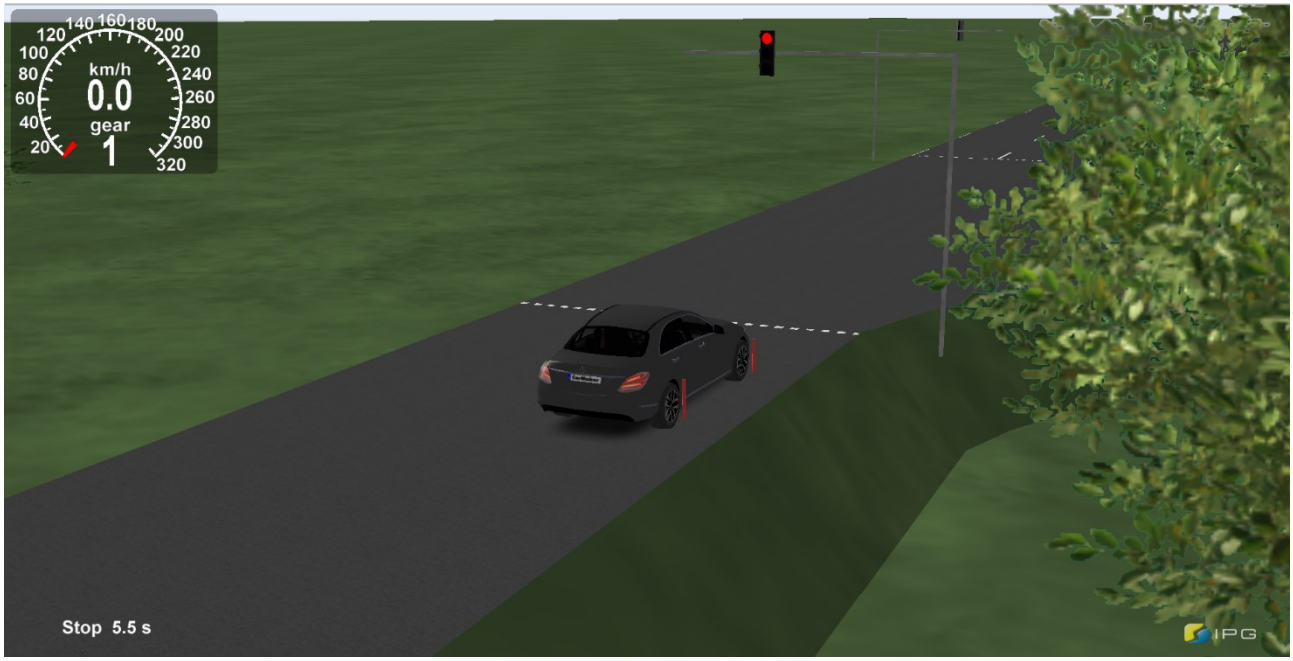


Figure 7-11: Legality check in open loop traffic light scenario

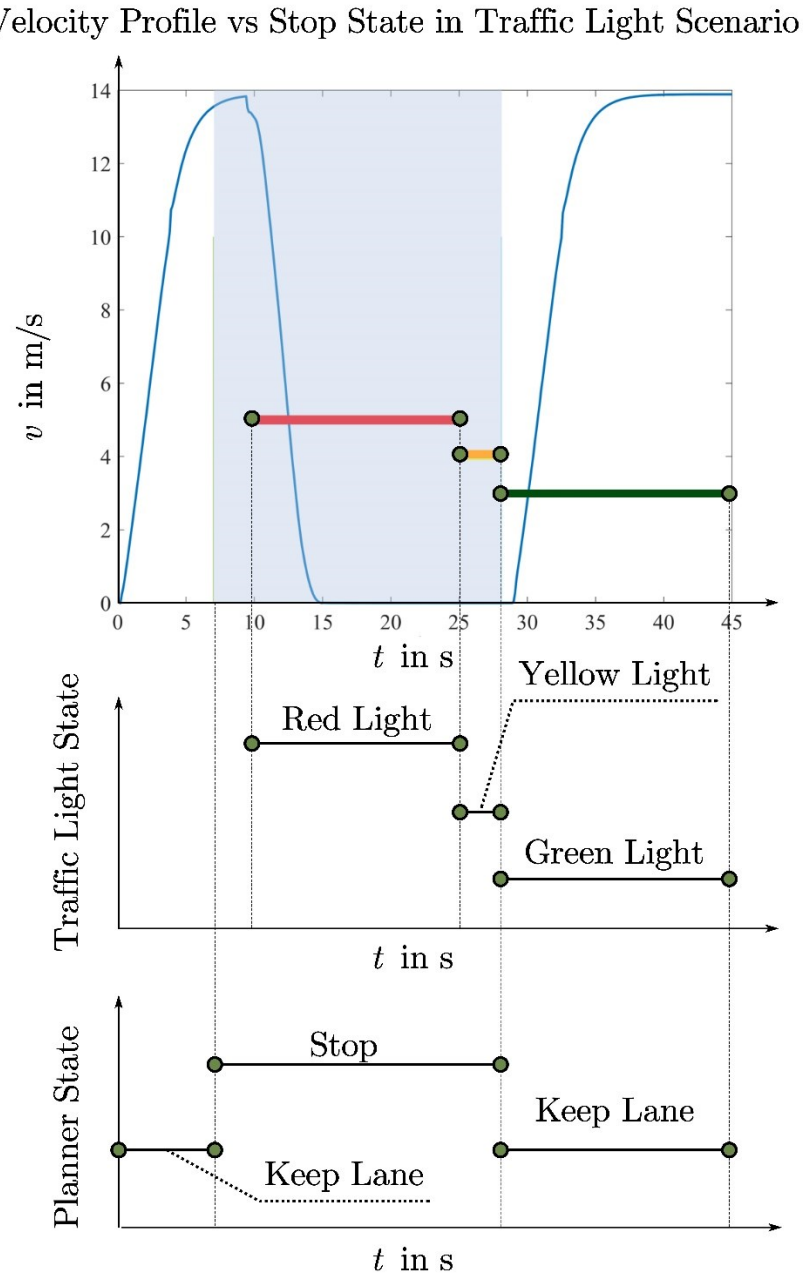


Figure 7-12: Legality of state in open-loop traffic light scenario

The red, yellow, and blue signals clearly show the traffic lights' changing process in the above picture. In the case of red light, as the vehicle approaches the intersection, the vehicle velocity also decreases, which is consistent with the vehicle speed profile. As mentioned in section 7-2, the traffic lights remain red during the first 25 seconds of the scene. As the ego car approaches, area recognition will also play a role. When entering the critical area, FSM will automatically jump to stop state. The light blue background represents the stop state command given by the behavior planner, which shows that the stop state is triggered earlier than ego car deceleration.

In summary, the correctness of state transition is successfully validated under open loop scenarios.

7.4.1.2 Correctness of Predicted Motion target

The correctness of the predicted motion target refers to the effectiveness of the motion target, which is closely related to the trajectory generation. As the boundary condition of trajectory generation, the motion target's error will directly affect the ego car dynamics' stability. Based on the recorded data, the most direct way to verify the correctness of the motion target is to compare it with the information of obstacles. As discussed in section 5.3.1, the final velocity of the motion target is always consistent with the speed of the obstacles. At the acceleration level, in order to avoid the generation of a jerk, the acceleration of the motion target is always zero, which means that the verification process must be performed at the displacement level.

The figure below displays the longitudinal displacement of obstacles and the motion target in the Frenet coordinate system.

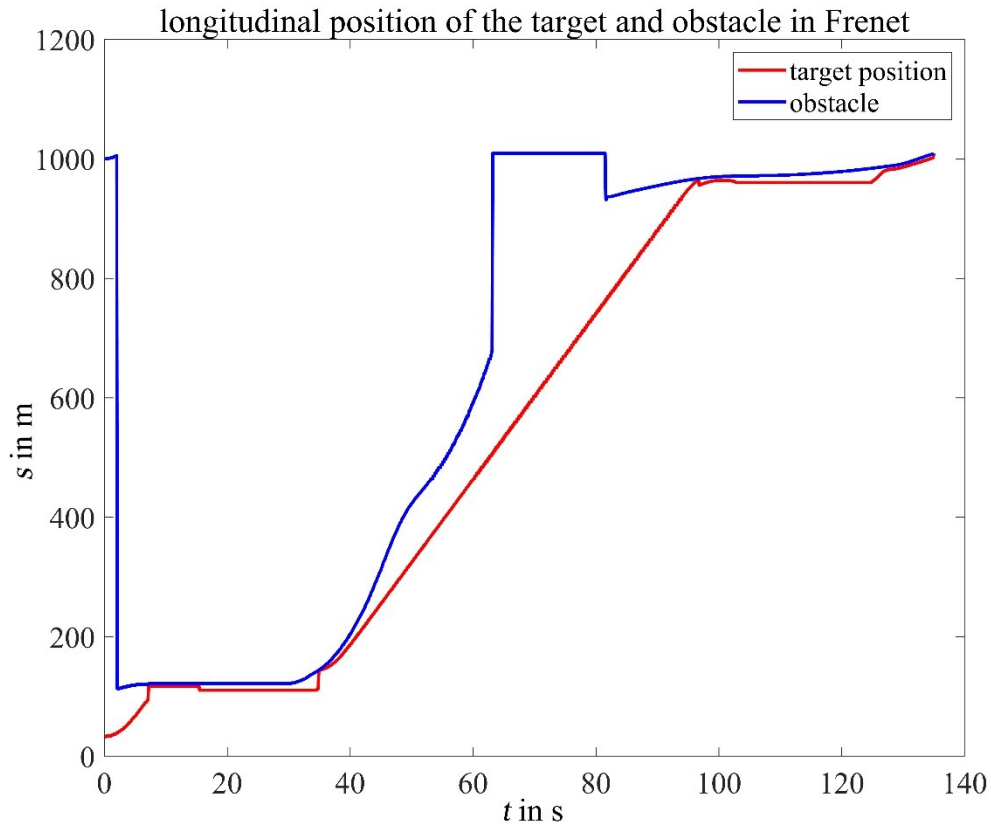


Figure 7-13: Motion target vs. obstacle in Frenet coordinate

In the figure, the blue line represents the longitudinal displacement of the obstacles, and the red line describes the longitudinal displacement of the motion target. Both of them are expressed in the Frenet coordinate system. Based on the default obstacles information mentioned in section 5.3.1, the following statistics can be obtained after filtering the active obstacles.

Table 7-3: statistics of relative distance between motion target and obstacles

obstacle minus motion target in meter			
mean	std	max	min
42.97 m	50.26 m	263.40 m	0.31m

As can be seen from the above, obstacles are always located ahead of the motion target in Frenet coordinate, and the minimum relative distance is still greater than zero. It is significant in selecting a motion target that makes the generation of trajectory avoid collision with the front obstacles.

7.4.2 Closed Loop Validation

Through the above discussion, the effectiveness of the behavior planner can be verified. The bottom of the decision-making layer, namely the trajectory planner, determines the stability of ego car vehicle dynamics and driving safety. This section is concerned with the validation of the trajectory planner, which works with behavior planner together in a closed-loop environment.

As the closed loop implies, the trajectory planner acts as a navigator. In this case, the built-in automatic algorithm logic of CarMaker will no longer work; contrarily, the trajectory planner will take over the dynamic control of ego car, such as velocity and orientation. The following IPG Movie screenshot below shows the ego car's under closed loop control operation in CarMaker.

Figure 7-14: Closed loop simulation in CarMaker

Since safety has a top priority in the validation process, it will be considered first in this section. As a criterion, TTC will play a major role in collision prediction and recognition. Consistent with the

previous section, in order to uniformly observe TTC changes during autonomous vehicle driving, TTCR is used here as a surrogate for TTC.

Figure below provides an overview of TTCR in traffic light scenario under closed loop environment.

Figure 7-15: inverse TTC under closed loop enviroment

As shown in above chart, the TTCR drops rapidly at about 30s. Because the relative distance cannot change quickly in a short time, the ego car has made an emergency brake at that moment. After braking, TTCR rebounded quickly. The maximum TTCR is about 0.8. After conversion, it can be concluded that the TTC in an emergency is about 4s. From the maximum of TTCR the ego car can always maintain a relatively safe distance from the obstacle in front. It shows that the urban autonomous framework has security that can be relied upon in the current scenario. According to the IPG Movie scene, the TTCR maintains a low level for a long time, which can be regarded as a constant speed cruise with no obstacles ahead. The sudden change of TTC means that the sensor captures new obstacles, which is closer to the ego car than the previous obstacles.

The following picture exhibits the comparison between the TTCR under closed loop and open loop. Excluding the influence of the autonomous driving algorithm, the two scenarios' parameters are identical: the traffic lights intersection with other traffic participants. In this case, the global planning of ego car is manually adjusted to drive straight through the intersection.

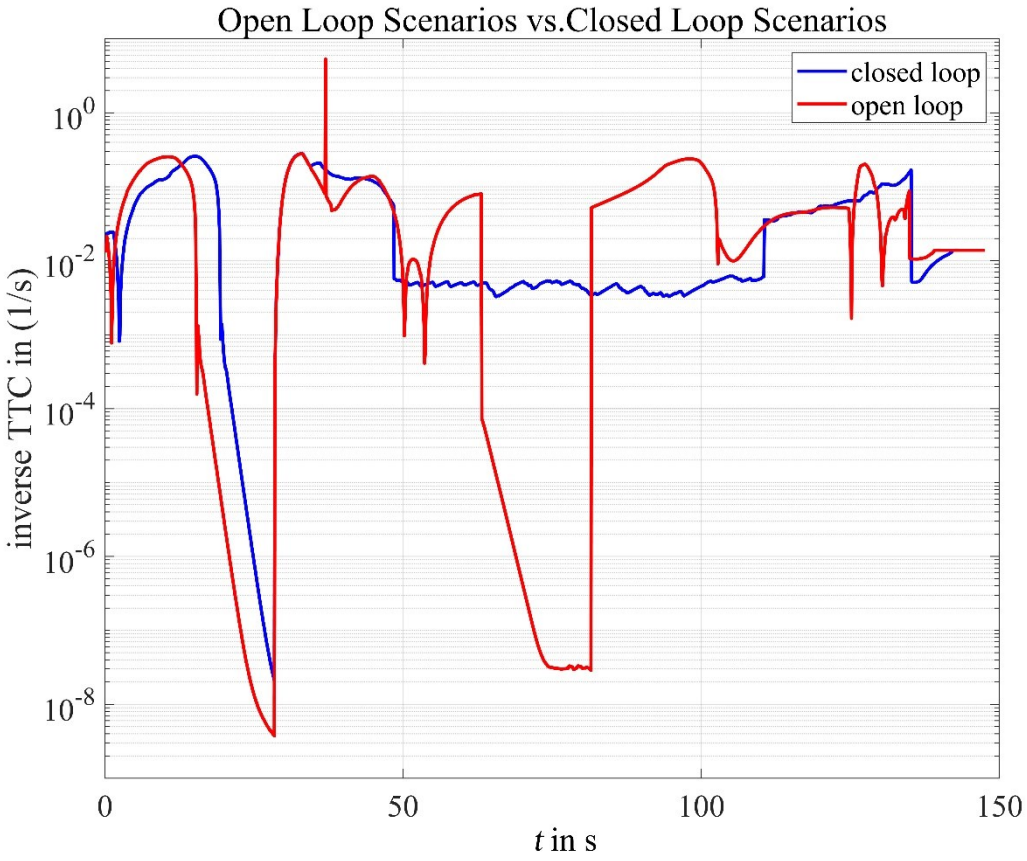


Figure 7-16: TTCR open loop vs. closed loop

The most interesting aspect of this graph is that the two TTCR curves are highly overlapping when obstacles appear ahead. It indicates that the autonomous driving framework in this project has achieved an excellent result, which is similar to Carmaker's built-in algorithm in terms of security. In the first 50 seconds of simulation time, the maximum TTCR of the closed loop appeared later than that of the open loop. It shows that, under the intersection scenario, the closed-loop control causes the ego car to have a different state of motion: the starting acceleration is slower. In this case, the TTCR of closed loop can still reach a similar level, which is sufficient to prove reliability and safety. With the behavior planner working smoothly and the trajectory planner operating normally, the urban autonomous driving framework has taken a substantial step towards realizing high-level autonomous driving.

8 Summary and Outlook

8.1 Summary of the project

The central purpose of the project is to design a behavior planner and a trajectory planner, which can meet self-driving demands. Both works smoothly individually. The behavior planner's design is based on the hybrid structure where the entire decision planning layer receives information from the upper layers, including environment perception, position and velocity of the ego car, and a global path planner.

The trajectory planner uses a quintic-polynomial and an optimal control approach to generate trajectories, which can meet the requirements of the limitation of the vehicle’s dynamic and the actual scenarios. A trajectory fitted by quintic-polynomial is jerk-optimal and smooth enough for the vehicle to track. The whole algorithm is based on the Frenet coordinate, so the transformation between the Frenet coordinate and Cartesian coordinate directly determines the result's quality. The cubic spline is smooth and convenient to calculate the curvature, tangential and normal vectors, and the reference line's arc length; hence it is used to fit the reference line in this project. Good fitting results have been tested in several demos in MATLAB and python files. After solving the necessary parameters needed in the transformation, the trajectory generation is not so complicated to complete in the next step. To generate a trajectory requires seven parameters: the positions, velocities, accelerations of the current ego car and target, and the required time. The method is quite sensitive to time, a wrong time parameter will lead the trajectory unacceptable, the trajectory planner still needs a lot of trainings to improve the parameter selection.

8.2 Outlook of the Project

8.2.1 Behavior Planner

In this project, the primary logical and operational modules by the behavior planner work correctly and take many different situations and requirements into consideration. However, it is still facing potential challenges:

- **Accurate prediction of obstacles collision**: so far, all the collision detection for blocks is performed in the vehicle coordinate system, which is exclusively suitable for straight driving scenarios. For steering obstacles located at the entrance of intersection, the collision check module cannot immediately identify potential collision risks. Therefore, it is necessary to implement the collision detection completely in Frenet coordinate system. In addition, the current prediction of obstacles and traffic participants is only dependent on the current detected motion states, which could lead to misjudgment by influence of perception noise. In order to improve the safety and efficiency, subsequent projects can establish a prediction model based on detection history and probability through a neural network to obtain more reliable traffic collision prediction.

- **Global planning in top-level layer**: the current behavior planner can only run appropriately with a pre-set global route. In the case of deviation from the established route, the behavior planner cannot re-plan the route information in real-time. Without a suitable global planning module, the whole framework will lack flexibility in the complicated scenario.

8.2.2 Trajectory Planner

As discussed in the coordinate transformation, the center line is the lifeline of the trajectory planner, if the preprocessing of the map generation is not precise enough, for example, as the Figure 8-1 shows, the tangential vector of previous and following road segments are not in same direction. Therefore, the cubic curve fitting of the center line in the trajectory planner will cause a series of problems, such like after cubic spline there are lots of "zic zac" curve in the center line. Due to the large curvature in the zic-zac segment, the calculation of trajectory could exceed the allowed maximal curvature of the autodynamics. To solve this problem, an extra check of these zic-zac should be conducted.

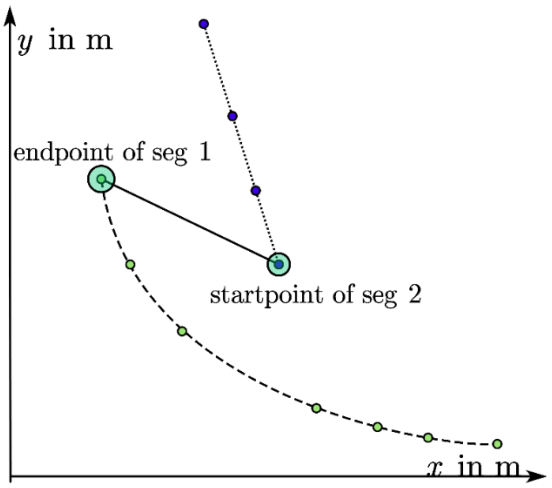


Figure 8-1: Zic-Zac curve in the map

Beside the "zic zac" issue, Fig 8-2 below shows how the sampling time influence the trajectory generation: two transient behaviors of the same planning strategy depending on the replanning frequency. A low replanning frequency causes overshoots or instability as the bottom trajectory planner indicates, due to violate the Bellman's principle of optimal, the optimal trajectory of the current state on the next iteration will be chosen that is slightly different.

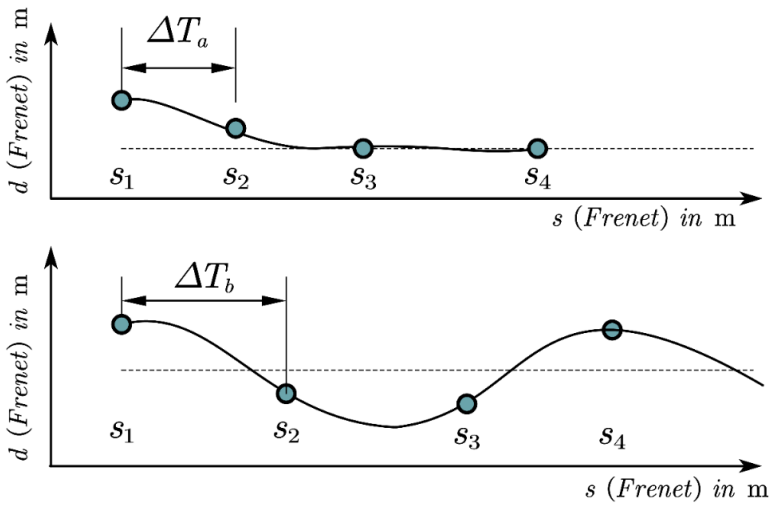


Figure 8-2: Low replanning frequency causes overshoots or instability

Moreover, the sampling time also has an impact on the start-up phase, when the car tries to start up again at the cross road. In our python demo, vehicle always possesses a relatively high velocity. However, in the close-loop simulation, when the vehicle changes its state from stop state to moving state, it takes a long time to achieve the target speed due to a small smapling time, a balance between the avoidance of overshooting and quick start-up in the further work should be found.

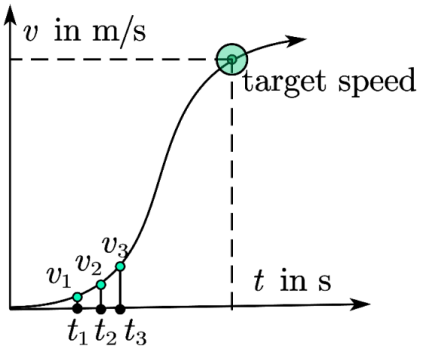


Figure 8-3: Velocity profile of the vehicle by Start-up phase

Appendix

List of References

A. Furda; L. Vlacic: Enabling Safe Autonomous Driving in Real-World City Traffic Using Multiple Criteria Decision Making (2011)
A. Furda; L. Vlacic: Enabling Safe Autonomous Driving in Real-World City Traffic Using Multiple Criteria Decision Making, in: IEEE Intelligent Transportation Systems Magazine (1), number 3, p. 4–17, 2011

A. Takahashi et al.: Local Path Planning And Motion Control For Agv In Positioning (1989)
A. Takahashi; T. Hongo; Y. Ninomiya; G. Sugimoto: Local Path Planning And Motion Control For Agv In Positioning, in: Proceedings. IEEE/RSJ International Workshop on Intelligent Robots and Systems '. (IROS '89) 'The Autonomous Mobile Robots and Its Applications, 1989

Baidu: ApolloAuto/Apollo
Baidu: ApolloAuto/apollo; https://github.com/ApolloAuto/apollo, accessed 05.02.2021

Clark, W. et al.: The Gantt Chart, a Working Tool of Management (1922)
Clark, Wallace; Polakov, Walter Nicholas; Trabold, Frank W: The Gantt chart, a working tool of management, The Ronald press company, New York, 1922

Donges, E.: Driver Behavior Models (2016)
Donges, Edmund: Driver Behavior Models, in: Winner; H. et al. (eds.): Handbook of Driver Assistance Systems: Basic Information, Components and Systems for Active Safety and Comfort, Springer International Publishing, Cham, 2016

Forsberg, K.; Mooz, H.: The Relationship of System Engineering to the Project Cycle (1991)
Forsberg, Kevin; Mooz, Harold: The Relationship of System Engineering to the Project Cycle, in: INCOSE International Symposium (1), number 1, p. 57–65, 1991

K. J. Lieberherr; I. M. Holland: Assuring Good Style for Object-Oriented Programs (1989)
K. J. Lieberherr; I. M. Holland: Assuring good style for object-oriented programs, in: IEEE Software (5), number 6, p. 38–48, 1989

Leitner, Andrea. et al.: Validation and Verification of Automated Systems : Results of the ENABLE-S3 Project. (2019)
Leitner, Andrea.; Watzenig, Daniel.; Ibanez-Guzman, Javier.: Validation and Verification of Automated Systems : Results of the ENABLE-S3 Project.; https://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=5975748, 2019,

Leonard, J. et al.: A Perception-Driven Autonomous Urban Vehicle (2008)
Leonard, John; How, Jonathan; Teller, Seth; Berger, Mitch; Campbell, Stefan; Fiore, Gaston; Fletcher, Luke; et al.: A perception-driven autonomous urban vehicle, in: Journal of Field Robotics (10), number 25, p. 727–774, 2008

M. Werling et al.: Optimal Trajectory Generation for Dynamic Street Scenarios in a Frenét Frame (2010)
M. Werling; J. Ziegler; S. Kammel; S. Thrun: Optimal trajectory generation for dynamic street scenarios in a Frenét Frame, in: 2010 IEEE International Conference on Robotics and Automation, 2010

Montemerlo, M. et al.: Junior: The Stanford Entry in the Urban Challenge (2008)
Montemerlo, Michael; Becker, Jan; Bhat, Suhrid; Dahlkamp, Hendrik; Dolgov, Dmitri; Ettinger, Scott; Haehnel, Dirk; et al.: Junior: The Stanford entry in the Urban Challenge, in: Journal of Field Robotics (9), number 25, p. 569–597, 2008

Moritz Werling et al.: Optimal Trajectory Generation for Dynamic Street Scenarios in a Frenét Frame (2010)
Moritz Werling; Julius Ziegler; S¨oren Kammel; Sebastian Thrun: Optimal trajectory generation for dynamic street scenarios in a Frenét Frame, in: ICRA 2010 number p. 987–993, 2010

Reinholtz, C. et al.: Odin: Team VictorTango's Entry in the DARPA Urban Challenge (2009)
Reinholtz, Charles; Hong, Dennis; Wicks, Al; Bacha, Andrew; Bauman, Cheryl; Faruque, Ruel; Fleming, Michael; et al.: Odin: Team VictorTango's Entry in the DARPA Urban Challenge, in: Buehler; M. et al. (eds.): The DARPA Urban Challenge: Autonomous Vehicles in City Traffic, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009