Computing Interaction-aware Reachable Sets of Automated Vehicles using Monte Carlo Tree Search

Scientific Thesis for the procurance of the degree M.Sc.
from the Department of Electrical and Computer Engineering at the Technical University of Munich.

Supervised by

Univ.-Prof. Dr.-Ing./Univ. Tokio habil. Martin Buss
M.Sc. Tommaso Benciolini
Chair of Automatic Control Engineering

Prof. Dr.-Ing. Matthias Althoff
M.Sc. Edmond Irani Liu
Chair of Robotics, Artificial Intelligence and Real-time Systems

Submitted by

cand. M.Sc. Xin Zhang
Sauerbruchstr. 59 0411
81377 Munich
049 - 1632653328

Submitted on Munich, 27.02.2023

TECHNISCHE UNIVERSITÄT MÜNCHEN
LEHRSTUHL FÜR STEUERUNGS- UND REGELUNGSTECHNIK
PRIV.-DOZ. DR.-ING. MARION LEIBOLD

February 27, 2023

M A S T E R ' S T H E S I S
for
Xin Zhang
Student ID 03739647, Degree EI

Computing Interaction-aware Reachable Sets of Automated Vehicles using Monte Carlo Tree Search

Problem description:

Highly-automated vehicles (AVs) promise an increase of road safety, time and cost efficiency compared to human drivers. Major challenges arise in dense, dynamic situations where the AVs should interact with human-driven vehicles. Game-theoretic approaches offer an elegant way tackle the interactions and dependencies between multiple agents. Based on certain assumptions, the ego agent can incorporate future interactions with other vehicles into its planning scheme. A decision process, such as using Monte Carlo tree search (MCTS), realizes the game theoretic setting by sampling primitive actions.
The aim of this thesis is to compute interaction-aware reachable sets of automated vehicles using Monte Carlo tree search. The over-approximative reachable set [1] of a vehicle is the set of states that can be reached by the vehicle over time, and encloses all its drivable trajectories. The computed reachable sets can then be used as planning constraints for subsequent motion planning of the vehicle. The benefit of computing the reachable sets instead of the direct trajectory is that it allows for adopting arbitrary motion planner in the subsequent stage, as well as offer the possibility to negotiate reachable sets for a group of explicitly cooperating vehicles.
The results should be demonstrated with CommonRoad [2] scenarios, an exemplary of which is shown below.

Tasks:

- Literature review on recent works on incorporating MCTS into motion planning.
- Familiarizing with CommonRoad and related software (reachable set computation, MCTS framework, etc.).
- Implementation of MCTS-based reachable set computation.
- Demonstration of computation results with various CommonRoad scenarios.
- Documentation of codes and other related materials.

Bibliography:

[1] S. Söntges and M. Althoff, "Computing the drivable area of autonomous road vehicles in dynamic road scenes," *IEEE Transactions on Intelligent Transportation Systems*, 2018.
[2] M. Althoff, M. Koschi, and S. Manzinger, "Commonroad: Composable benchmarks for motion planning on roads," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017.

Supervisor:	Tommaso Benciolini, M.Sc. and Edmond Irani Liu, M.Sc.
Start:	01.08.2022
Intermediate Report:	XX.XX.XXXX
Delivery:	28.03.2023

(M. Leibold)
Priv.-Doz.

Abstract

Autonomous driving technology has played a key role in the development of motorized transportation. In dense and dynamic scenarios, the self-driving car should interact with human-driven vehicles, which is one of the major challenges for ego vehicles to complete autonomous driving tasks safely and efficiently. We use a game-theoretic approach to address the interactions and dependencies between multiple agents. The motion planning problem of self-driving cars is described as a single-player game, and decision planning is performed by using the single-player Monte Carlo tree search (SP-MCTS). The motion planner can effectively find the feasible path with the assistance of the reachable set in a dense traffic scenario. The action system of SP-MCTS for autonomous vehicles is defined based on the reachable set. We assume that the orientation of the vehicle is always consistent with the reference path, simplify the definition of the action, and slice the acceleration interval into primitive actions. To realize the interaction between the vehicle and the surrounding vehicles, vehicles will be categorized in the SP-MCTS simulation stage based on the relatively safe distance, and sequential decisions will be made based on the influence relationship. The ego vehicle uses heuristic random selection, which considers the information of the road and the surrounding vehicles, to determine actions in the simulation. The reachable set obtained by SP-MCTS is called the interaction-aware reachable set, which contains high-level maneuvering information of the ego vehicle. The interaction-aware reachable sets are beneficial for subsequent motion planning and offer the possibility to negotiate reachable sets for a group of explicitly cooperating vehicles. The project is implemented through CommonRoad Benchmark [AKM17], and the algorithm is verified with various CommonRoad scenarios.

Contents

Chapter 1

Introduction

Autonomous vehicles are a rapidly developing technology that could improve the safety, accessibility, efficiency, and convenience of motorized transportation. The requirements for autonomous vehicles to handle complex situations are also increasing, especially for decision-making capabilities. To solve the complex decision-making problem of autonomous driving, the article [PČY+16] proposed to divide the decision-making problem into four layers. The first is route planning based on the road network. We can convert the road network into a direct graph, and then plan a route from its current position to the requested destination by finding the minimum cost path. As a next step, the vehicle makes high-level decisions based on the surrounding environment and traffic participants, also known as maneuver planning. As soon as the maneuver has been confirmed, such as a lane change to the right, we need to translate it into a trajectory, which should be dynamically feasible for the vehicle, avoid collisions with obstacles, and be comfortable for the passengers. In the final step, control the vehicle so that it tracks the determined trajectory.

In dense and dynamic scenarios, the decision-making planning of self-driving cars should consider the interaction with human-driven vehicles. Monte Carlo tree search is often used to solve sequential decision problems [ŚGSM22], and can simulate the interaction between multiple agents and dependencies. Therefore, we use MCTS for maneuver planning by sampling primitive actions. The over-approximative reachable set proposed by [SA17] of a vehicle is the set of collision-free states that can be reached over time starting from an initial set of states, and encloses all drivable trajectories. With reachable sets, search spaces are explored in continuous space more efficiently and the feasible trajectory can be detected even in narrow passages[SA17]. It is possible to solve the dense traffic path planning problem efficiently using the reachable set [WA21]. And it allows arbitrary motion planners to be employed in subsequent stages, as opposed to computing the trajectory directly. So we define primitive actions based on reachable set. Then we use MCTS to calculate the sequential action chain, and we can get the interaction-aware reachable set through the action chain. The interaction-aware reachable set can express the maneuvering strategy of the ego vehicle under interaction with other vehicles.

1.1 Problem Statement

In this section, we will introduce the problems that this project aims to solve as well as the difficulties that will arise in the implementation process. As shown in figure 1.1, the gray area is the over-approximative reachable set at $2s$ of the ego vehicle in an intersection. For the convenience of expression in the following, the over-approximated reachable set is simply called the reachable set. This reachable set can be roughly separated into two parts according to the status of the ego vehicle at $2s$ if we perform path planning based on reachable sets. If the state of the ego vehicle is inside the red box, the trajectory of the ego vehicle should cross the intersection before the yellow vehicle. The ego vehicle will pass through the intersection after the yellow car if it is in the area enclosed by the green box. The ego vehicle needs to answer the question of which of the two maneuvers to choose. The driving logic of human drivers is, for example, if the yellow vehicle is approaching an intersection quickly, we give priority to letting it pass. And if the yellow vehicle is far away and the speed not high, we will consider passing the intersection first. The above analysis shows that the decision logic of the ego vehicle should be based on interacting with the yellow vehicle. We hope to calculate the interaction-aware reachable sets through MCTS, which can well solve the interaction problem between the ego vehicle and the human-driver vehicle. And interaction-aware reachable sets can express a driving strategy.

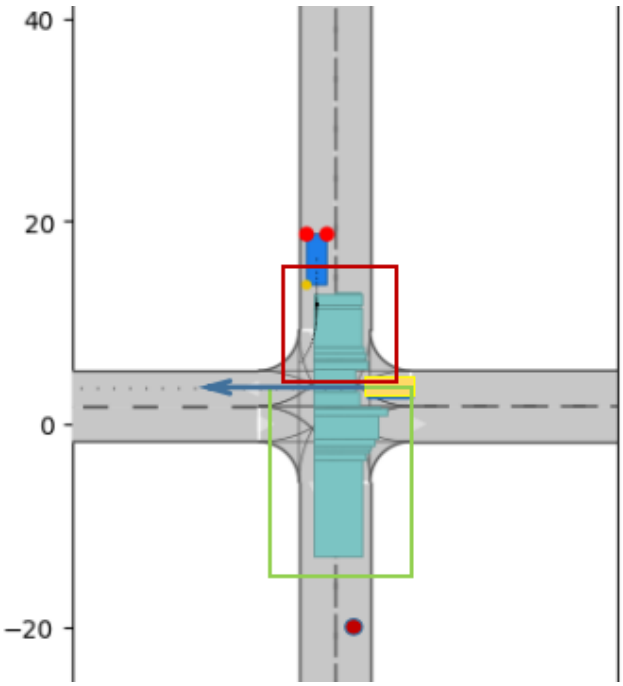


Figure 1.1: Reachable sets at 2 s in scenario ZAM_Intersection-1_2_T-1, red point denotes the initial position of ego vehicle and yellow block represents the human-driven vehicle from right to left

The main problem we need to solve in the implementation process is how to construct a logical framework for the interaction between the ego vehicle and the surroundings vehicles, and determine what kind of behavior model to use to simulate and predict the behavior of the human-driver vehicle. The surrounding vehicles are driven by humans, so we cannot control them to achieve cooperative behavior in real scenario. Vehicles only interact in the simulation stage of MCTS (see figure 2.1). MCTS is a tree search that solves problems by sampling primitive actions, hence the action space is discrete. But the reachable set is expressed in a continuous space. We need to establish an appropriate definition for an primitive action based on the reachable set, and an evaluation system.

1.2 Related Work

Reachable Sets for Trajectory Planning

The paper[GPMN15] provides an overview of the various motion planning approaches developed in the past. The algorithm relevant to our project are discretization-based approaches, such as fast exploration of random trees [KF11] or state lattices [MUDL11], which generate paths by sampling in a kinematically feasible way. They require exhaustive sampling of the state space in order to preserve reactive properties [GPMN15], which leads to increased computation time of the planner. Such approaches are notoriously difficult in obstacle fields with tunnels, or tubes. The exploration process of RRT based on reachable set is proposed in paper [SWT09], which greatly improves the performance of the algorithm in motion planning. In addition, set-based reachability analysis and convex optimization are combined to find driving actions even in small and complex solution spaces [MPA20]. Planning often generates high-dimensional search problems that must be solved under limited computational resources and difficult real-time constraints. It is mentioned in the article [FHL08] that the problem of trajectory generation is always transformed into a graph search problem. This graph-based search problem is prone to the problem that the size of the graph increases exponentially with respect to the number of search dimensions. The reachable set can significantly reduce the search space of the search method by considering the possible search space of static and dynamic obstacles [SA17]. In the article [WA21], it is proposed to prune the search space of sampling-based motion planners by reachable set. By creating samples only in the collision-free reachable set, the number of required samples is greatly reduced.

Monte Carlo Tree Search for Motion Planing:

In the article [LKK16] , 4 advantages of MCTS in the field of automatic driving are summarized.

- Anytime: MCTS can always be stopped and a result is available. It might not be optimal but it is valid.
- Parallel: MCTS can be highly parallelized, either many iterations at once, or multiple simulations in one iteration.

- Cooperative/Adversarial: MCTS can generate cooperative or adversarial behaviors depending on the cost functions.
- Versatile: MCTS can account for different planning strategies for different traffic participants (Each node and each action can have different implementations).

In paper [BEHK20], they built BARK, an open-source behavioral benchmarking environment designed in which behavioral models are used for planning, prediction, and simulation. The library gathers such as Intelligent Driver Model, Mobile Model, and various behavioral models based on reinforcement learning, and they proposed a single-agent MCTS in which other agents are assumed to follow a certain modeled behavior. Algorithms designed for single-agent systems tend to suffer from the curse of dimension when applied to multi-agent systems. The number of actions grows exponentially with the number of agents, which is aggravated when planning for longer periods [AAA23]. This paper [LKK16] proposed that a multi-agent mobile traffic participant considers the interaction between different vehicles to plan a cooperative motion plan by defining information set [CPW12] to complete simultaneous decision-making of vehicles, and there is no communication between vehicles. While the algorithm proposed in the paper[KZZ18] is based on cooperative modeling of other agents and decoupled UCT (a variant of MCTS [TLW14]), evaluates the state-action value of each agent in a cooperative and decentralized manner, explicitly modeling the traffic between participants behavioral interdependence.
The search space of path planning problem is generally continuous, but the problem solved by MCTS should be discrete. Therefore, the subject of how to define action is important. In article [LKK16], each action corresponds to a value (acceleration), and the macro-action system is defined in [KZZ18], which contains some primitive actions, and each primitive action corresponds to a single behavior. In this paper[KEZ18], collaborative modeling based on other agents and decoupled UCT is also used. The uses the definition of semi-action group [SWUVDH07] to group action spaces. And in the article[ZA23], the state space of a kinematic single-track model is discretized by using the speed-dependent steering angle transformation. For a detailed analysis of the definition of action space, please refer to section 3.3.1.

1.3 Structure

The remainder of this article is organized as follows. In this chapter 2, we will introduce all of the principles involved in this project, including game theory, Monte Carlo tree search, reachable set, and auto behavior model (Intelligent Driver Model). In chapter 3, we will focus on the framework of the algorithm and details of implementing the algorithm. In chapter 4, a total of 7 scenarios will be used to evaluate the performance of the algorithm and we will analyze and explain the simulation results. Chapter 5 briefly summarizes the performance of the algorithm according to the simulation results, discusses the aspects of the algorithm that are worth optimizing, and proposes some methods that can improve the algorithm. Finally, chapter 6 conducts a comprehensive review of the project and look forward to future work.

Chapter 2

Preliminaries

2.1 CommonRoad

The hardware and software used in this project are discussed in this section. First of all, the code development and testing in the project are based on the CommonRoad platform, which is led by Prof. Althoff and is used to develop motion planning for self-driving cars. The platform is mainly developed in python language, which contains many libraries for the development and verification of autonomous driving technology.

- CommoRoad-IO: It mainly implements the parsing and generation of scenario files in XML format. The scenario includes a lanelet-based road network system and various types of obstacles, such as static and dynamic objects. Vehicles are generally defined as dynamic objects, which record the geometric information (like length and width) and initial state of the object, as well as the trajectory over time. In addition, the package provides convenient visualization tools, including scene building tools that can be built and constructed by the traffic simulator SUMO Build.
- CommoRoad Drivability Checker: This toolbox can check the drivability of a trajectory, complete collisions among obstacles detection, and include conversion functions from Cartesian coordinates to curved coordinates.
- CommonRoad Route Planner: CommonRoad Route Planner is used to define reference path and offer motion-planning algorithms high-level assistance.
- CommonRoad-Reach: CommonRoad-Reach unifies various approaches for calculating reachable sets and extracting driving corridors for automated vehicles in dynamic traffic scenarios.

The library activation and usage tutorial mentioned above can be found at *commonroad.in.tum.de*. The development of this project is all based on a computer with 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GH. This computer is installed with Ubuntu 18.04, and we use Pycharm as the primariy development tool. To avoid compatibility issues between versions, we recommend using Ubuntu 20.04 or 18.04.

2.2 Game Theory

Game theoretic reasoning pervades economic theory and is used widely in other social and behavioral sciences[Gib97], such as game development and the self-driving industry. Game theory can help decision makers make decisions considering interaction, so game theory can play a very important role in decision-making planning for autonomous driving, for example, [LW20] construct game theory lane change strategy. Here are two very important theories in game theory:
1. *Rational Choice Theory*: The action chosen by a decision-maker is at least as good as every other available action, not according to preferences.
Theories using rational choice models aim to derive meanings that do not depend on any qualitative characteristics of preferences. In practical situations, the driving behavior of the vehicle is related to the driver. Regardless of whether the driver's style is aggressive or conservative, we assume that traffic participants do not make decisions based on preferences. The purpose of the assumption is that there are no so-called preferences in the decision-making process, which facilitates the decision-making process. Select the most reasonable plan from them, and help to establish a mathematics-based evaluation system to the greatest extent.
2. *Nash Equilibrium*: A Nash equilibrium is an action profile a^* with the property that no player i can do better by choosing an action different from a_i^* , given that every other player j adheres to a_j^*. [Gib97]
When all participants make decisions based on the theory of the rational choice model, the Nash equilibrium point will eventually be reached, and when there is no pressure to change actions, no one will change their actions unless others change their actions. Reaching the balance point also requires the beliefs of players about actions each other to be correct, which is sometimes interpreted as the expectations are coordinated of player [Gib97]. For example, in urban traffic, there will be regulations on " rechts vor links " at intersections in Germany, and this regulation requires the consent of all participants. Drivers also have higher priority when passing the intersection when they are on the right.
Games can be classified by the following features, and the explanation of features comes from [BPW+12].

- Zero-sum: Whether the reward to all players sums to zero (in the two-player case, whether players are in strict competition with each other).
- Information: Whether the state of the game is fully or partially observable to the players.
- Determinism: Whether chance factors play a part (also known as completeness, i.e. uncertainty over rewards).
- Sequential: Whether actions are applied sequentially or simultaneously.
- Discrete: Whether actions are discrete or applied in real-time.

MCTS works best in this type of combinatorial game. Combinatorial game is a type of two-player game, which is zero-sum, perfect information, deterministic, discrete, and sequential,

such as Go, and chess. Therefore, the self-driving problem model constructed in this project tries to meet these features. In the article [Cha10], the basic requirements for using MCTS are summarized. MCTS is applicable if at least the following three conditions are satisfied: (1) the payoffs are bounded(the game scores are bounded), (2) the rules are known (complete information), and (3) simulations terminate relatively fast (game length is limited).

2.3 Monte Carlo Tree Search

2.3.1 Structure of Monte-Carlo Tree Search

Monte Carlo tree search (MCTS) was proposed by Remi Coulom [Cou07b] to be used in Crazy Stone, a Go-playing program. After AlphaGo defeated lee sedol in 2016 and Ke Jie in 2017, two top Go human players, MCTS became very famous because of its role in AlphaZero of Google DeepMind. Various variants of MCTS have been developed in the field of games and have been extended to different fields. Monte Carlo tree search is a general approach to solving sequential decision-making problems under uncertainty using stochastic (Monte Carlo) simulation. With the rapid development of autonomous driving in recent years, MCTS is also more and more for motion planning of autonomous vehicles.

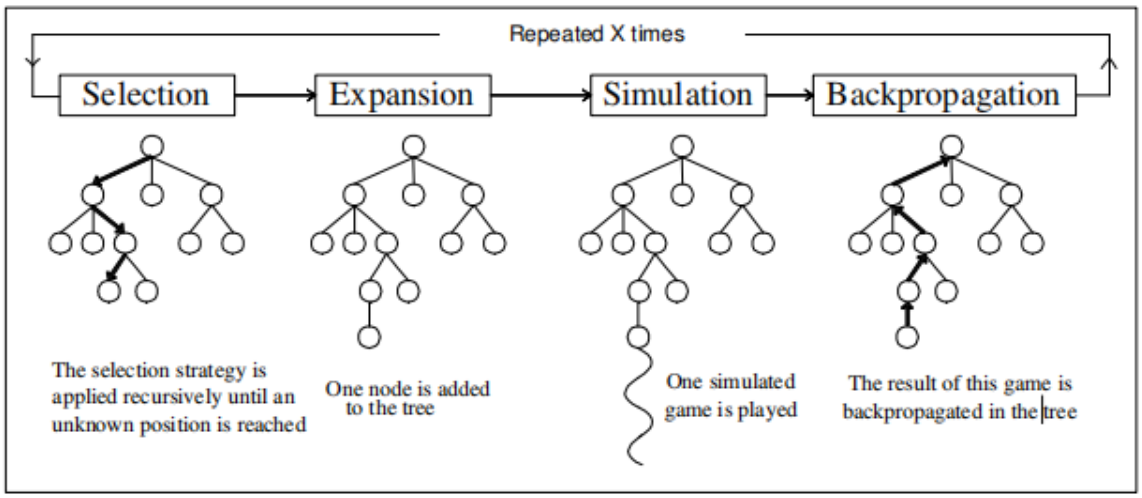


Figure 2.1: Outline of Monte-Carlo Tree Search (cited from [Cha10])

MCTS is a type of tree search algorithm. There are two important parts in a tree search problem: the nodes of the tree, also called states, and the branches of the tree, which are the one-way edges connecting two nodes, also called actions. A search tree is defined as an acyclic-connected graph in which each node has a set of zero or more child nodes and at most one parent node. Each node holds at least two values, the number of times it has been visited and a value that corresponds to the total reward of all playouts that passed through this state. The meaning of this reward value is generally defined according to the problem. The average

of the results of simulated games that visited the node is used in most circumstances. Figure 2.1 shows the structural flow of MCTS. Within the specified calculation time, MCTS starts from the root node and repeats four steps: selection, expansion, simulation, backpropagation. In general, we will refer to selection and expansion as tree policy, and simulation is also known as default policy. There is a pseudo-code 1 that can help with understanding. In the selection step, the tree will travel from the root node to the leaf node. A leaf node is a node with no children. The next step is to expand the new child node to join the tree from the leaf node. Then start the simulation from the expanded new node until the end of the game, that is, reach the terminal node and return a reward value. This reward value will be propagated back to the root node through new child nodes in the backpropagation step. Finally, we will select best one of the child nodes of the root node as the next move according to a certain rule.

Algorithm 1 Pseudo-code for Monte-Carlo Tree Search

Input: RootNode
Output: BestChildNode

```
1: while TimeBudget do
2:     newChildnode ← TreePolice(RootNode)
3:     Reward ← DefaultPolice(newChildnode)
4:     Backpropogation(Reward)
5: end while
6: return BestChildNode ← FinalMoveSelection(RootNode)
7:
8: function Tree Police(CurrentNode)
9:     Node ← CurrentNode
10:    while Node is not LeafNode do
11:        Node ← Select(Node)
12:    end while
13:    NewChildNode ← Expand(LeafNode ← Node)
14:    CurrentNode.addChild(NewChildNode)
15:    return NewChildNode
16: end function
17:
18: function Default Police(NewChildNode)
19:    Reward, TerminalNode ← Simulate(NewChildNode)
20:    return Reward
21: end function
22:
23: function Backpropagation(CurrentNode, Reward)
24:    while CurrentNode.parent do
25:        CurrentNode ← CurrentNode.parent
26:        CurrentNode ← Backpropagate(Reward)
27:    end while
28: end function
```

Through the above, we can understand from a high-level perspective what the MCTS algorithm tree does at each stage, and how to search for the best action. The next few sections will introduce each stage in detail.

2.3.2 Selection

In the selection step, the tree starts from the root node and recursively applies the selection strategy to select the child node until reaching a leaf node. The selection strategy needs to be able to balance exploitation and exploration. First, the task requires the ability to choose the action (exploitation) that brings the best result so far, which is what we use MCTS for. But because of the uncertainty, it still has to try the less promising action due to uncertainty assessment (exploration).
Next, we will show the most used algorithm Upper Confidence Bounds for Trees (UCT) for balancing exploitation and exploration in MCTS, and briefly explain how to achieve the balance through it. Let $\mathcal{N}_c(N)$ be the set of child nodes of the current node N. UCT selects a child node N_k of the node N that satisfies formula 2.1:

$$N_k \in argmax_{N_i \in \mathcal{N}_c(N)} \left(\frac{v_{N_i}}{n_{N_i}} + C \times \sqrt{\frac{2In(n_N)}{n_{N_i}}} \right) \tag{2.1}$$

where v_{N_i} is the value of the node N, n_{N_i} is the visit count of N, and n_N is the visit count of N. C is a positiv coefficient. The first term in the formula is referred to as exploitation, while the second term is considered to as exploration. C can be adjusted to lower or increase the amount of exploration performed. The setting of this value needs to be adjusted according to the specific situation through experiments.
When n_{N_i} equals zero the resulting UCT value is infinity, so previously unvisited children are assigned the largest possible value to ensure that all children of a node are visited at least once before any children are further expanded. As each node is visited, the denominator of the exploration term increases, which makes the exploration term smaller and reduces its impact. On the other hand, if another child node of the parent node is visited, the numerator increases, so the exploration value of the unvisited sibling node increases. The exploration term ensures that each child has a non-zero probability of selection, which is essential given the randomness of the playout. This also gives the algorithm an inherent restart property, as even children with low rewards will eventually be selected (given enough time) and thus can explore different lines of play [BPW+12].

Final Move Selection

After MCTS finishes using the computation budget, we need to select the best child node according to a certain standard from all child nodes of the root node, that is, the optimal action. This is completely different from the purpose of selecting child nodes in the selection step. In the article [CWH+08], four methods of selecting the optimal child node are summarized through parameters such as the number of visits and the value of the node.

1. *Max child*: The max child is the child that has the highest value.

2. *Robust child*: The robust child is the child with the highest visit count.

3. *Robust-max child*: The robust-max child is the child with both the highest visit count and the highest value. If there is no robust-max child at the moment, more simulations are played until a robust-max child is obtained[Cou07b].

4. *Secure child*: The secure child is the child that maximizes a lower confidence bound.

In the experiments with MANGO, there was no significant difference between the above methods when a sufficient number of simulations were performed per move [Cha10]. The final method chosen by our project is *Max child*.

2.3.3 Expansion

After the selection step, the tree search reaches the leaf nodes. A leaf node has no children and is a non-terminal node. Then an available action can be selected from the action space to expand the child node of the leaf node. Alternatively, because a certain limited search depth has been reached, the reached node will be defined as a leaf node. Such a leaf node has a child node but is not fully expanded. That is, some nodes can still be expanded. To expand, we need to choose one of those nodes at random that has not been visited before. When the leaf node is fully expanded, we need to execute selection instead of expansion. In practice, some additional constraints are given to filter expanded nodes. For example, in a maneuver planning task, if the maneuver of the vehicle is to accelerate, but a collision will occur, we will skip this action directly and choose those actions that will not cause a collision. The most commonly used strategy in this step is only one child node is added per expansion. The strategy at this step has only a small impact on the final simulation results and creating one node per simulation is therefore sufficient in most cases.

2.3.4 Simulation

In the simulation step, the search tree will start from the node obtained by the tree policy and perform self-playing according to the default policy until the game (system) ends, or reaches the terminal node. The simulation strategy (default) can generally be divided into the random selection and pseudo-random moves chosen. Using an appropriate simulation strategy can significantly improve the effectiveness of the MCTS algorithm.
There are two trade-offs that should be considered for a simulation strategy [Cha10]. The trade-off between search and knowledge is one of them, which can be seen as a trade-off between computational cost and algorithm performance. Adding knowledge to simulated strategies can make self-playing more logical and cognitive. The more accurate the simulated game becomes, the more reliable the results of its simulation. However, if the computational cost of the heuristic knowledge is too high, the time required for the simulation will increase dramatically. The trade-off between exploration and exploitation is another one. If the policy is too random,

too much exploration will be done. The actions performed by the agent during the simulation are often illogical, resulting in an unrealistic simulation and a reduction in the level of the Monte Carlo program. For example, when the speed of the vehicle in front is very low, the agent should not accelerate, but because of random selection, the agent randomly chooses to accelerate so that the two vehicles collide. The simulation ends early. Conversely, too much exploitation can occur if the policy is also too deterministic. Exploration of the search space becomes too selective, leading to biased simulations and reduced levels of Monte Carlo tree search.

2.3.5 Backpropagation

Backpropagation is to assign the result (reward) of the simulated game to the node selected by the tree police and backpropagate from the node to the root node. In this process, all nodes from the root node to the leaf node will be traversed and updated with the number of visits and value of the node. Comparing various backup strategies in the document [Cou07a], we can conclude that the current relatively good strategy is to use *Average*, which takes the common average of all simulation game results made through this node.

2.3.6 Single-Player Monte-Carlo Tree Search

Many of the variants of MCTS are summarized in [ŚGSM22], [BPW+12], and the references therein. According to the characteristics of each variant of MCTS, it is finally decided to use single-player MCTS (SP-MCTS) in our work. Chapter 3 will show a detailed analysis of why this variant was chosen. This section will mainly introduce SP-MCTS based on paper [SWVDH+08]. The emphasis is on the distinction between SP-MCTS and standard MCTS.
[SWVDH+08] proposed to build a variant of MCTS to solve single-player games with perfect information. It mainly solves the strict requirements for an accurate admissible evaluation function of Classical methods such as A^* and IDA^* [Kor85], which are popular and successful choices for one-player games. The main differences between it and the standard MCTS are in the selection and backpropagation stage.
The adapted UCT in SP-MCTS is organized as:

$$\bar{X} + C \times \sqrt{\frac{Innn_N}{n_{N_i}}} + \sqrt{\frac{\sum v^2 - n_{N_i}\bar{X}^2 + D}{n_{N_i}}} \tag{2.2}$$

The first two terms are still the original UCT formula 2.1. It uses the number visits n_N of node N and the number of visits n_{N_i} of children N_i to give a confidence upper bound on the average game value of $\bar{X}$ [SWVDH+08]. The third term is added, which represents a possible child node deviation. It contains the sum of squared results $\sum v^2$ corrected by the expected result $n_{N_i}\bar{X}^2$ obtained so far in the child nodes. A high constant D is added to ensure that rarely explored nodes are considered indeterminate [SWVDH+08].
First of all, the essential difference between a single-player game and a two-player game based on the cumulative point system lies in the range of values of the node. In a two-player game,

the outcome of the game is explicitly lost, drawn, or win, i.e. $\{-1, 0, 1\}$. The average value of a single node can be kept in the range $[-1, 1]$. In a cumulative single-player game, the average value of nodes can be any value and will not be limited to a certain range. As a solution [SWVDH+08] to this problem, we can set the constants (C, D) to be feasible for a range of values. The second solution is to shrink the values back to the above interval $[-1, 1]$, which requires a given maximum score. A theoretical upper bound can be used when the exact maximum score is not known. For example, in SameGame, the theoretical bound is to assume that all blocks have the same color. A direct consequence of such a high upper bound is that the game score will be located near zero. This means that in a single-player game, the constants C and D are set to completely different values compared to a two-player game[SWVDH+08]. In the backpropagation stage, the simulation results at the leaf nodes are propagated back to the root node. In the backpropagation stage of SP-MCTS, the average score of a node will be updated, which is the same as MCTS. In addition to this, SP-MCTS also updates the sum of squares of the results (see equation 2.2) due to the third term in the selection strategy, and the best score obtained so far due to computational reasons.

2.4 Auto Behavior Model

For the simulation in the virtual environment, we need some logical cognitive vehicle behavior models to help us obtain reasonable simulation effects, especially for the game between interactive vehicles. This article [BEHK20] described various vehicle behavior models. The Intelligent Driver Model (IDM) is a vehicle-following model often used in microscopic traffic simulation, which will also be implemented in this article. And a MOBIL model is a decision-making entity to trigger lane changes to improve traffic flow, which adds lane change judgment based on the IDM model. Reinforcement learning is used for behavior control by collecting experiences and updating its policy to maximize the long-term cumulative expected reward, primarily through the use of the Proximal-Policy-Optimization (PPO) and the Soft-Actor Critic (SAC) algorithm. In the article [LKK16], the behavior control of the vehicle is completed by using MCTS to select an action in the action set, which is called Multi-Agent Monte Carlo Tree Search Model. In our project, we will realize that ego vehicles use MCTS for behavior control and other vehicles use IDM for behavior control.

$$a_{IDM} = \frac{dv_\alpha}{dt} = a\left[1 - \left(\frac{v_\alpha}{v_0}\right)^\delta - \left(\frac{s^*(v_\alpha, \Delta v_\alpha)}{s_\alpha}\right)^2\right] \tag{2.3}$$

$$s^*(v_\alpha, \Delta v_\alpha) = s_0 + v_\alpha\, T + \frac{v_\alpha\, \Delta v_\alpha}{2\sqrt{a\, b}} \tag{2.4}$$

$$\dot{v}_\alpha^{\text{free}} = a\left(1 - \left(\frac{v_\alpha}{v_0}\right)^\delta\right) \qquad \dot{v}_\alpha^{\text{int}} = -a\left(\frac{s^*(v_\alpha, \Delta v_\alpha)}{s_\alpha}\right)^2 \tag{2.5}$$

The acceleration of IDM in equation 2.3 can be divided into free road acceleration $\dot{v}_\alpha^{\text{free}}$ and deceleration parts $\dot{v}_\alpha^{\text{int}}$. The free road acceleration $\dot{v}_\alpha^{\text{free}}$ represents the acceleration of the ve-

hicle from the current speed to the desired speed. The added exponent δ represents how the acceleration will decrease as the velocity increases. For example, when the exponent δ is equal to 1, the acceleration decreases linearly. As the exponent approaches infinity, the acceleration is constant and equal to the maximum acceleration a. The addition of the deceleration component $\dot{v}_\alpha^{\text{int}}$ causes the current vehicle to react to the behavior of the leader vehicle. Equation 2.4 represents the effective minimum clearance s^* between the current vehicle and the leader vehicle. If the distance between the current vehicle and the vehicle in front is not significantly greater than the effective minimum clearance, the deceleration will not become relevant. The effective minimum gap is characterized by the minimum distance s_0 (which is relevant for low velocities only), the velocity-dependent distance $v_\alpha T$ which corresponds to following the leading vehicle with a constant desired time gap T, and a dynamic contribution which is only active in non-stationary traffic corresponding to situations in which Δv is not equal 0 [KTH10].

Under normal circumstances, the acceleration of the vehicle will be reduced to a comfortable level. When the distance between the two vehicles is very close or the vehicle behind is easily catching up to the vehicle in front, the second part will assist the vehicle in increasing deceleration, preventing both vehicles from colliding. Each parameter will be organized in a table 2.1 to make it easier to query its meaning.

Table 2.1: Parameter Description for IDM

Parameter	**Explanation**
v_0	desired velocity
δ	free acceleration exponent
a	maximum acceleration
b	comfortable deceleration
T	desired time gap
s_0	minimum desired safe distance
Δv	approaching rate

2.5 Reachable Set

The calculation of over-approximate reachable set in this project is based on previous works [SA17], [LWKA22]. This chapter will briefly introduce definition of the reachable set and how to calculate one-step over-approximate reachable set. In the field of autonomous driving, a reachable set is defined as a set containing all states that satisfy dynamic constraints (velocity and acceleration bounds) and do not collide with the external environment. The calculated reachable has three application directions: trajectory planning and trigger driving assistant systems, in particular collision mitigation systems and reducing assess the risk of a traffic situation [SA17]. The paper [SA17] proposed an over-approximation algorithm to calculate the reachable set more efficiently in a dynamic environment. The computation flow chart of over-approximate reachable set is shown below in figure 2.2.

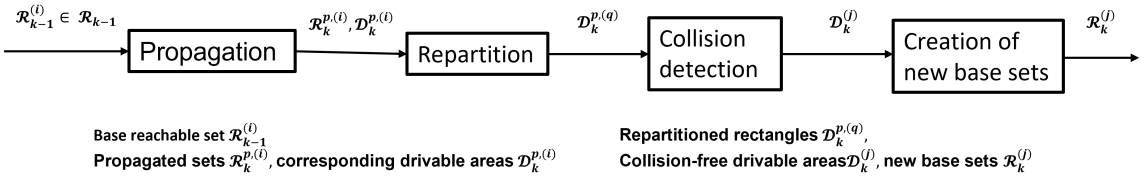


Figure 2.2: Computation of reachable set

The following will show the mathematical definition of reachable set. Trajectory of the state expression is:

$$s(t) = s_0 + \int_{t_0}^{t} f(s(\tau), u(\tau)) d\tau \tag{2.6}$$

s_0 is the initial state, $u(\tau)$ is the input signal and $\frac{ds}{dt} = f(s(t), u(t))$ is the system dynamic. It can use the state equation to express the relationship between position velocity and acceleration in 2 dimension platform, as well as the hard constraint of velocity and acceleration.

$$\begin{aligned} \begin{pmatrix} \dot{x} \\ \ddot{x} \\ \dot{y} \\ \ddot{y} \end{pmatrix} &= \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} u_x \\ u_y \end{pmatrix} \\ |u_x| &\leq a_{max,x} \\ |u_y| &\leq a_{max,y} \\ v_{min,x} \leq \dot{x} &\leq v_{max,x} \\ v_{min,y} \leq \dot{y} &\leq v_{max,y} \end{aligned} \tag{2.7}$$

The reachable set can be defined as the set of all states that can be reached from an initial set $\mathcal{X}_0$ at time t:

$$\begin{aligned} \mathcal{R}(t; \mathcal{X}_0) = \{ s(t; u, s_0) \mid & \exists u \in \mathcal{U}, \exists s_0 \in \mathcal{X}_0, \\ & s(\tau; u, s_0) \notin \mathcal{F}(\tau) \, for \; \tau \in [t_0, t] \} \end{aligned} \tag{2.8}$$

$\mathcal{F}$ is a set of forbidden states, which is assumed to be given. In the expression 2.7, the state of the vehicle is divided into x directions and y directions, which are independent of each other. An over-approximation of the reachable set of the joined two-dimensional motions is shaped by several merged reachable sets of one-dimensional motions. The mathematical solution for the reachable set of the one-dimensional motion along the x direction will be demonstrated.

To calculate the reachable set in the x direction, we can only consider the acceleration limit first and then add the speed limit. We assume that t_i is the time step and $\mathcal{I}_i$ is the position interval at t_i. The system model in the x direction can be seen in equation 2.9.

$$\begin{aligned}\begin{pmatrix}\dot{x}\\ \ddot{x}\end{pmatrix} =& A\begin{pmatrix}x\\ \dot{x}\end{pmatrix} + Bu_x\\ A = \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}&, B = \begin{pmatrix}0\\ 1\end{pmatrix}\\ |u_x| \leq a_{max,x}&\\ x(t_i) \in I_i&\end{aligned} \tag{2.9}$$

Then, a more detailed reachable set without any position constraints is given by formula 2.10. The character $\oplus$ denotes the Minkowski sum. $\mathcal{P}_{u,x}(t)$ is the set of all reachable states from an initial state with all possible acceleration constrained inputs.

$$\begin{aligned}\mathcal{R}(t;\mathcal{X}_0) &= \left\{s \mid \exists u \in \mathcal{U}, \exists s_0 \in \mathcal{X}_0, s = e^{At}s_0 + \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau\right\}\\ &= e^{At}\mathcal{X}_0 \oplus \left\{s \mid \exists u \in \mathcal{U}, s = \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau\right\}\\ &= e^{At}\mathcal{X}_0 \oplus P_{u,x}(t)\\ &= \left\{a + b \mid a \in e^{At}\mathcal{X}_0, b \in P_{u,x}(t)\right\}\end{aligned} \tag{2.10}$$

We can use optimal control theory to determine the maximum and minimum speeds in specific terminal position x_t at time t, and use Pontryagin′s principle to obtain a bang-bang input candidate function with switching time γt. We can get upper velocity bound (full braking until time γt, then full acceleration) and lower velocity bound (full acceleration until time γt, then full braking) in equation 2.11. When we take $\gamma = 0,\ 0.5,\ 1$ for switching times, we can get a polytope approximation using supporting halfspaces. The result is shown in figure 2.3

$$\begin{aligned}x_t^{(h)}(\gamma) &= x_0 + \dot{x}_0 t + a_0 t^2(\frac{1}{2} - 2\gamma + \gamma^2)\\ \dot{x}_t^{(h)}(\gamma) &= \dot{x}_0 + a_{max,x}(1 - 2\gamma)\\ x_t^{(l)}(\gamma) &= x_0 + \dot{x}_0 t - a_0 t^2(\frac{1}{2} - 2\gamma + \gamma^2)\\ \dot{x}_t^{(l)}(\gamma) &= \dot{x}_0 - a_{max,x}(1 - 2\gamma)\end{aligned} \tag{2.11}$$

After that, the calculation of the reachable set also needs to add a speed constraint to get all possible states $\mathcal{P}_i$ at each time step in equation 2.12, that is, the intersection of the propagated one-time step set and the allowed position interval with speed constraints $[v_{min,x}, v_{max,x}]$.

$$\begin{aligned}\mathcal{P}_i \subseteq &\left(e^{A(t_i - t_{i-1})}\mathcal{P}_{i-1} \oplus \mathcal{P}_{u,x}(t_i - t_{i-1})\right)\\ &\cap (\mathcal{I}_i \times [v_{min,x}, v_{max,x}])\end{aligned} \tag{2.12}$$

The equation 2.12 is an over-approximation, because the speed constraint may be violated in between the time interval [SA17]. Each base reachable set $\mathcal{R}_i$ is considered as a Cartesian product of two convex polytopes $\mathcal{P}_{x,i}$ and $\mathcal{P}_{y,i}$ in x and y direction [LWKA22].

We can calculate the basic reachable set by giving the upper- and lower-bound of acceleration as well as velocity. We can also obtain the corresponding reachable set if we slice the acceleration intervals $[a_{min}, a_{max}]$. If there is an acceleration slice sequence over time corresponding to a maneuver used to calculate reachable sets, the resulting reachable sets can show corresponding maneuvers. The continuous acceleration interval $[a_{min}, a_{max}]$ constitutes the MCTS search space in our project. Primitive actions are also defined based on this acceleration range.

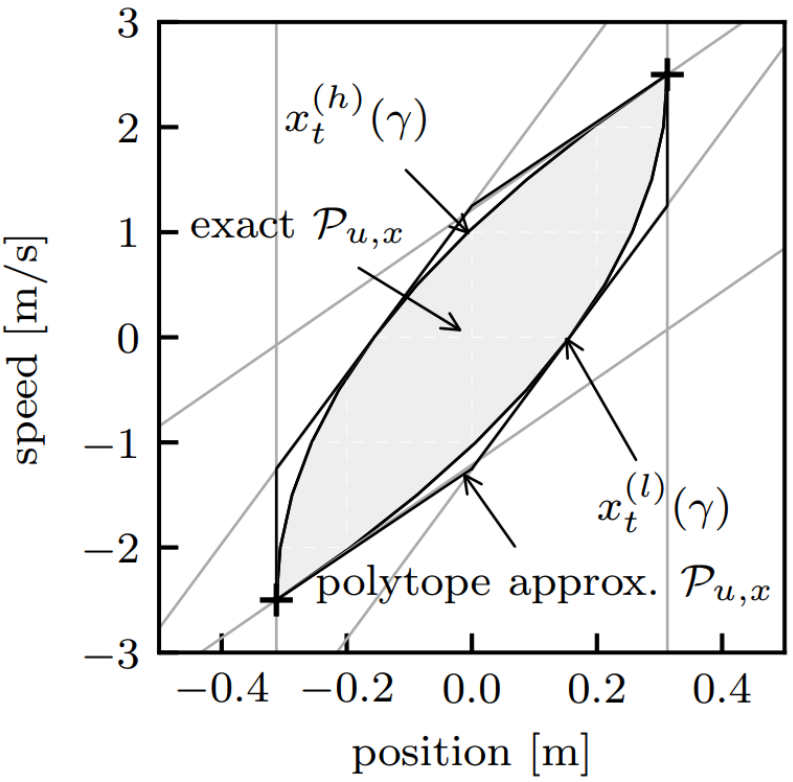


Figure 2.3: Closed-form solution $\mathcal{P}_{u,x}$ of the reachable set of an acceleration bounded point mass with initial state $[0, 0]^T$ and a polytope approximation (cited from [SA17])

Chapter 3

Technical Approach

3.1 Overview of Algorithm

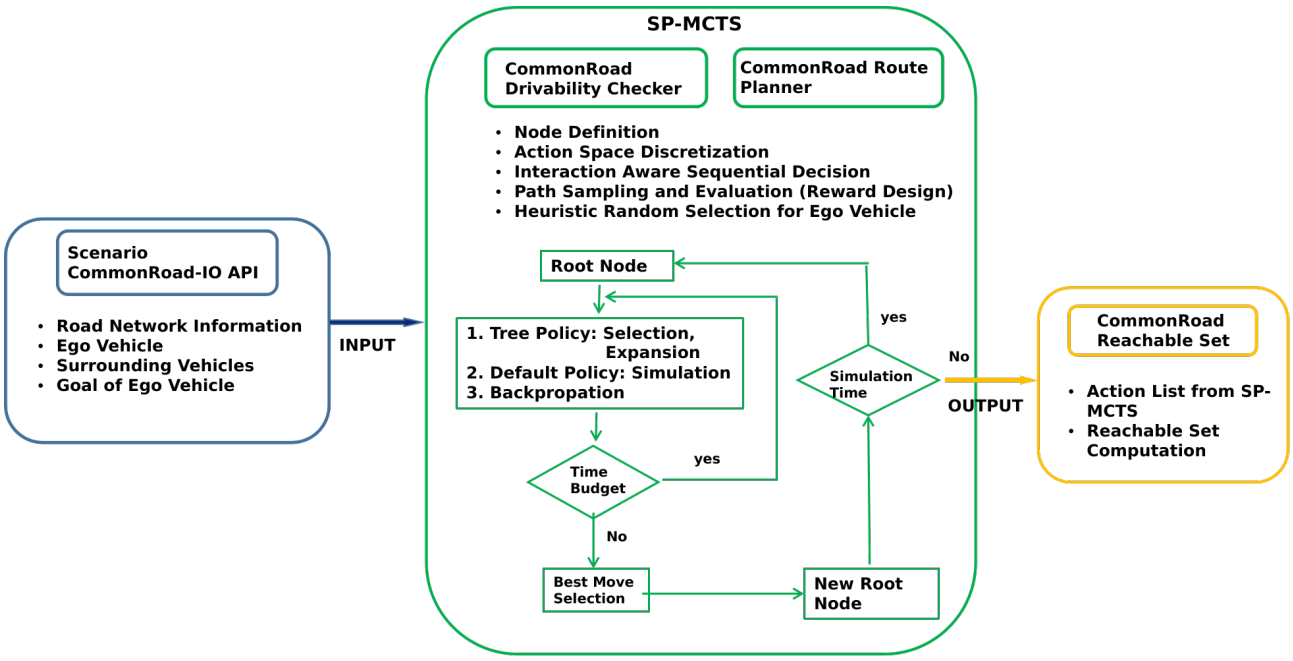


Figure 3.1: Overview of algorithm structure

In this chapter, we will explain briefly how the computation of the interaction-aware reachable set based on CommonRoad is achieved using single-player MCTS. The overview of algorithm structure is depicted in figure 3.1. The road network information and the basic information of the vehicle are parsed from the scenario through the CommonRoad-IO API, including the kinematics and geometric information of the vehicle and the goal of planning problem of the ego vehicle. We will construct the MCTS node and a plausible primitive action system using this information. After that, SP-MCTS will be used to finish the user-defined length of time T_s simulation and search out a list of acceleration intervals (primitive action). The acceleration

interval list will be used to compute interaction-aware reachable sets via the CommonRoad-Reach API. For example, if we set the simulation time T as 3s and the discretization time t as 0.1s, then SP-MCTS will return a list with a length of 30, where each element is an acceleration interval. In SP-MCTS, expansion, selection, and interaction-aware simulation need to be completed. We will develop a sequential decision framework for simulations that are interaction aware. The ego vehicle is controlled by the action system, while the other vehicles are controlled by the IDM or maneuver according to some assumption. Additionally, a continuous space-based reasonable evaluation method for action is required.

3.2 Traffic System

3.2.1 Road Network

Localization in high-definition maps is a key problem for autonomous navigation since vehicles need to extract information from them [HMXB17]. In CommonRoad benchmarks, lanelets are used as atomic, interconnected, and drivable road segments to represent the road network. It is powerful enough to fulfill all major needs in driving simulators and automated driving [AUK18]. A lanelet is established by its left and right bounds, which are represented as an array of points (a polyline), as figure 3.2. The use of lanelets makes it possible to describe the road network as a directed graph, each node having four types of outgoing edges: successor, predecessor, adjacent left, and adjacent right [AKM17].

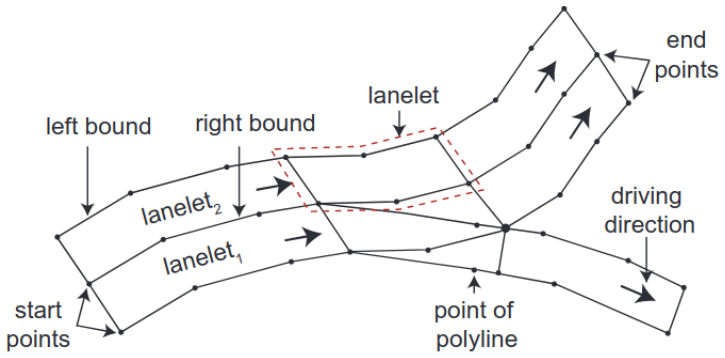
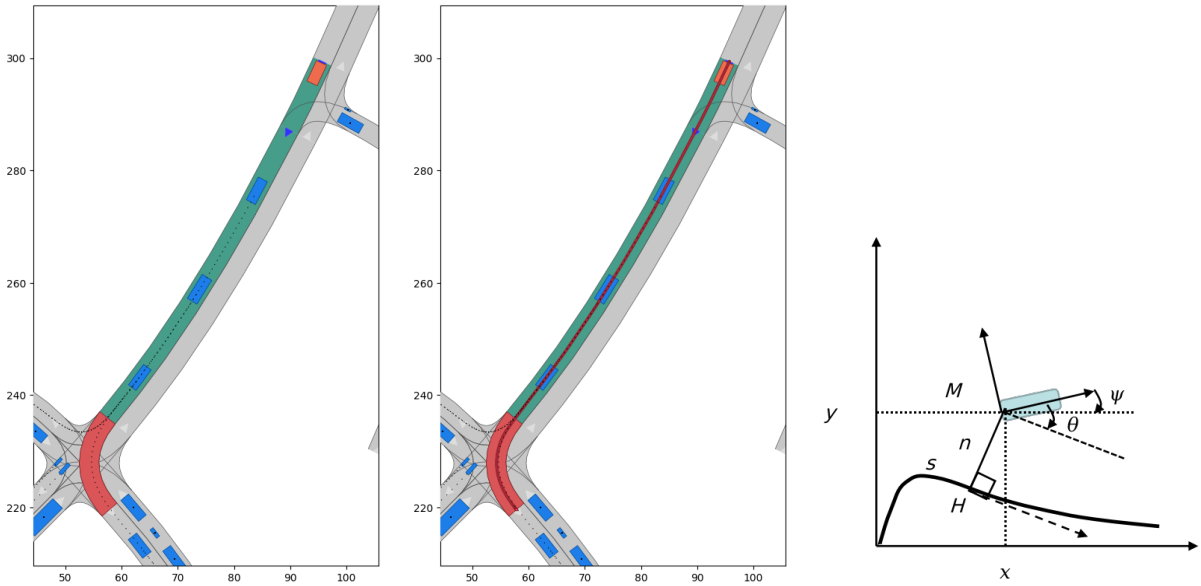


Figure 3.2: Description of the lanelet [AUK18]

If we know the starting and the ending lanelet, then we can plan a reference lanelets path through the directed graph, as shown in the figure 3.3. A reference trajectory can be calculated by using CommonRoad Route Planner API and a reference curve coordinate system with the reference trajectory will be built in our project. Reference path and the curvilinear coordinate system will contribute to path evaluation and vehicle state update. In the Cartesian coordinate system, the pose of the vehicle can be expressed as $[x, y, \theta]$. We can convert it to the curved coordinates $[n, s, \psi]$ with the curvilinear abscissa s, the lateral distance n from the reference trajectory, and the angle ψ between the orientation of the vehicle θ and the angle of the tangent

of the reference trajectory. In the follow-up work, the ego vehicle will be assumed to follow the reference trajectory, so the orientation of the ego vehicle is the same as the angle of the tangent of the reference trajectory. The relative orientation ψ will be ignored in the follow-up work and in order to avoid unclear meanings of letters, we use $[p_{lat}, p_{lon}]$ ($[p_x, p_y]$) to replace $[n, s]$.



(a) Reference lanelets path (b) Reference trajectory (c) Curvilinear coordinate system

Figure 3.3: Reference Path and curvilinear coordinate system

3.2.2 Auto Model

The vehicle model is used to update the state of the vehicle and specifies the kinematics of the vehicle. The document [AW20] provides a summary of a number of car models, as well as diversions and detailed descriptions. As we see in the illustration 3.4, the single-track kinematics model is used in our project. Here only an introduction to the parameters is provided without in-depth analysis.

In figure 3.4, l_{wb} describes the wheelbase. δ is the steering angle and v_δ the velocity of steering angle. a_{long} is the acceleration in the longitudinal direction, which is the same as velocity v. The differential equations of the kinematic single-track model are

$$\begin{aligned} \dot{\delta} &= v_\delta, \\ \dot{\theta} &= \frac{v}{l_{wb}} tan(\delta), \\ \dot{v} &= a_{long}, \\ \dot{p}_x &= v cos(\theta), \\ \dot{p}_y &= v sin(\theta) \end{aligned} \tag{3.1}$$

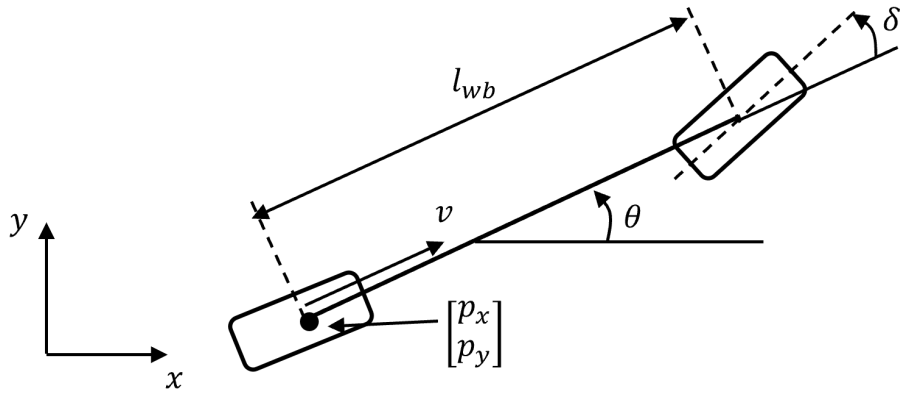


Figure 3.4: Single-track kinematics model

3.2.3 Game Model

We can translate the problems we face into game descriptions. In the project, the ego vehicle is the only player, and it knows all the information about the road and other traffic participants. It needs to get a higher score by reaching the goal as much as possible within the given time. The goal includes the goal and desired velocity. The vehicle plays the game by selecting primitive actions from our defined action set. The score of the vehicle will depend on how well the task is completed and whether there is a collision. This game is a single-player game with non-zero-sum, perfect information, deterministic, continuous, and sequential. The standard MCTS does not handle this single-player game very well, as stated in the section 2.2. We discover single-player MCTS, a variation of MCTS that is better suited for single-player games.
The parameters of the vehicle and the basic MCTS node we built are given in the following table 3.1. The state of the vehicle is a tuple, including (a_{long}, v, p, θ). A node records the current simulation duration t_N to determine whether it is a terminal node. Action set $\mathcal{A}(N)$ and untried action set $\mathcal{A}_{untried}$ are used in the expansion stage. N_p and $\mathcal{N}_c$ manage the parent node and child node set. In the backpropagation phase, the visit number n_N and value v_N of the node will be updated. When the search is done, the a_{list} will be output for computing the reachable set.

3.3 Single-Player Monte Carlo Tree Search

In the article [BPW+12], the author created a thorough summary and analysis of MCTS and its variations. In the field of autonomous driving, in addition to the standard MCTS, multi-agent MCTS is the variant of MCTS that is most frequently used. Because of the interaction between vehicles, the action set of each vehicle will be joint, resulting in a huge search space. MCTS was developed to solve the huge search space problem in the Go game, so multi-agent MCTS makes contributions in this field. In multi-agent MCTS, instead of a single agent competing with itself in the simulation phase, the effects of having multiple agents (i.e., multiple simulation strategies) are considered. The bursty nature of the interactions between different agent types

Table 3.1: Parameter description for ego vehicle and node N in MCTS

Parameter	Description
	Ego Vehicle Information
a_{long}	acceleration in longitudinal direction
v	velocity of vehicle
p	position $[x, y]$ of vehicle in cartesian coordinate system
θ	orientation of vehicle
a_{max}	maximum acceleration in longitudinal direction (7 m/s^2)
a_{min}	minimum acceleration in longitudinal direction (-6 m/s^2)
	Node N Information
$\mathcal{A}(N)$	set of all available actions for this node N, $\mathcal{A}(N) = \{a_1, a_2, ...a_n\}$ and $a_i = [a_{i,min}, a_{i,max}]$
t_N	simulation time, length of a_{list} is $\frac{t_N}{\Delta t}$ with $\Delta t = 0.1s$
a_{list}	list of history action from root node N_r to current node N
a_t	executed action in node N
$\mathcal{A}_{untried}$	set of untried action, $\mathcal{A}_{untried} \subseteq \mathcal{A}(N)$
N_p	parent node of node N, $N_p = N.parent$
$\mathcal{N}_c$	expanded child node, $\mathcal{N}_c = \{N_{c,i} \mid N = N_{c,i}.parent\}$
n_N	the visit number of node N
v_N	value of node N

leads to the increased exploration of the search space[SM11]. However, finding an agent set with the correct properties is computationally intensive[BPW+12]. And multi-agent MCTS is decision-making for cooperative behavior, which means that not only ego vehicles, but also other vehicles, are required to make decisions. For example, in the decoupled UCT used in the article [AAA23], multiple agents make decisions at the same time but remove dependencies on the decisions of others. In this project, the ego vehicle needs to complete the task and be controlled by action, and other vehicles cannot be controlled. We consider only non-ego vehicle behavior control for prediction in the simulation phase of MCTS. The method used in our project is for a single agent, the cooperation of multiple agents is accomplished by sequential decision-making of the vehicle during the simulation phase. During the simulation process, the decision for each vehicle is made sequentially after the other, and the simulation is run for a fixed time after the last decision.
According to the game rules we devised, ego vehicles must complete the goal of reaching the destination as much as possible in order to obtain higher game scores. This is a straightforward definition of a single-player game. So we try to use SP-MCTS to complete the action list search.

3.3.1 Action Space

The goal of our project is to find a small continuous acceleration segment $[a_l, a_r]$ on a large continuous action space $[a_{min}, a_{max}]$ every time MCTS is executed. Before explaining the methods used in this project, we list 4 methods used in other articles. In the article [LKK16], the

action space it builds is a set of primitive actions. Each primitive action represents a single value as shown in figure 3.5. The majority of projects use this definition of action space, but it is insufficient for this project's needs. During the course of this project, this action space definition was applied to test whether SP-MCTS can effectively complete motion planning.

Action	*Preconditions*	*Description*
a_{i,v_0}	-	Keep the current velocity constant
$a_{i,v_0,\pm}$	-	Increase or Decrease the current velocity with a fixed acceleration a_{acc}
$a_{i,T}$	Leading vehicle	Keep time gap constant to leading vehicle
$a_{i,\circledS}$	Stopping point	Come to a stop at a stopping location. This could be the end of lane, an intersection entering, an obstacle ahead, ...
$a_{i,\rightleftarrows}$	Lane left/right	Sets reference lane parameter to the left or right lane to perform a lane change

Figure 3.5: Action space with single-value primitive action, cited from [LKK16]

The article [KZZ18] proposed to build a macro action space(MAs) first, and each macro action contains a subset of primitive actions. The primitive actions are defined as acceleration(+), deceleration (-), do-nothing(0), lane change to left (L), and lane change to right (R). They are represented by quintic polynomials describing [THNS89] the changes a set of p discrete speed changes in longitude $\{\Delta\dot{x}_k, k = 1 \cdots p\}$ and lateral position $\Delta y \in \{\Delta d_l, 0, \Delta d_r\}$. The problem of cooperative planning with MAs will be formulated as a decentralized Semi-Markov Decision Process(Dec-SMDP).

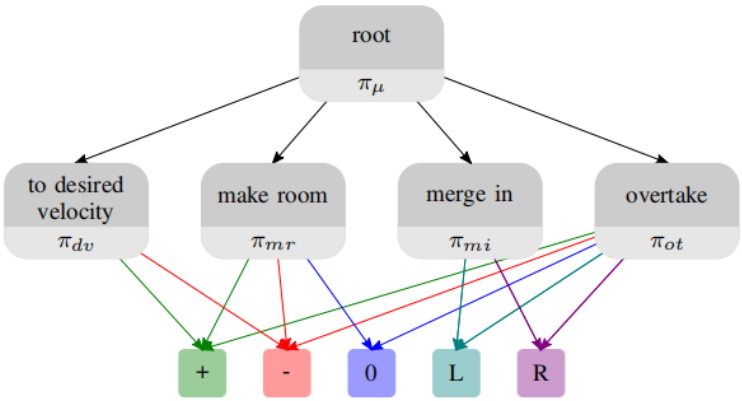


Figure 3.6: Macro action space (cited from [KZZ18]) with primitive actions: acceleration(+), deceleration (-), do-nothing(0), lane change to left (L), and lane change to right (R)

In the article [KEZ18], an action is also defined as a pair of values with a longitudinal velocity change and a lateral position change. And proposed Semantic Action Grouping to reduce the computational complexity and increase the robustness of the algorithm. Semantic action groups

describe state-dependent, discrete areas within the action space of an agent. The action space is divided into a total of nine areas in the lateral and longitude directions. In the lateral direction is Lane change to right (R), lane keep(0) and lane change to left(L), and in the longitude direction acceleration(+), constant speed(0), and deceleration(-).

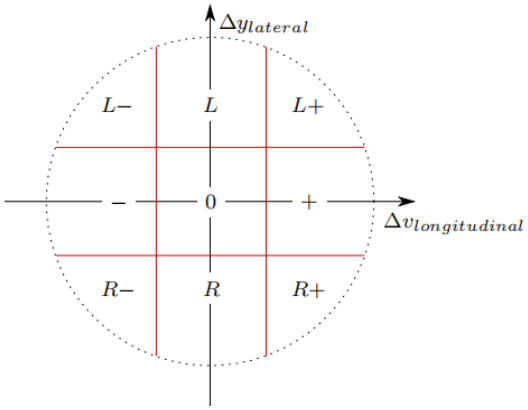


Figure 3.7: Semantic action group (cited from [KEZ18]), L- means the agent change lane to left and decelerate in longitudinal direction

In this article [ZA23], the self-driving vehicle uses a single-track model to ensure that only physically possible trajectories will be planned, so the composition of the action is an acceleration value as well as a steering rate. According to the discretization distance Δa, the minimum acceleration a_{min} and the maximum acceleration a_{max}, the set of discrete acceleration values is calculated as

$$a_d \in \{a_{min} \leq p\Delta a \leq a_{max} \mid p \in \mathbb{Z}\} \tag{3.2}$$

With proper selection of δa, some common behavior patterns can be represented, such as emergency braking, comfortable braking, coasting down, sustained speed, comfortable acceleration, and emergency acceleration. The steering rate in action is the speed-dependent steering angle transformation from used. δ_{max} is the maximum drivable path curvature, $a_{lat,max}$ is the maximum lateral acceleration, and l is the wheelbase, v is the speed calculated by the discrete acceleration of the vehicle.

$$|\delta_{max}| = min(arcsin(\kappa_{max} l), arcsin(\frac{a_{lat,max} l}{v^2(a_d)})) \tag{3.3}$$

We use the kinematic single-track model to translate the state of the vehicle over time. In theory, one primitive action in our project should incorporate both the acceleration in longitudinal direction and the steering rate. We assumed the vehicle will follow the reference trajectory planned by CommonRoad Route Planner, like figure 3.3(b) , the orientation of the vehicle is same as the angle of the tangent of the reference trajectory. Only the longitudinal acceleration needs to be taken into account. The available longitudinal acceleration is a continuous interval. The action space will be discretized in the project based on the potential behavioral pattern it

can represent. Here, the action space is divided into uniform speed, comfortable acceleration, rapid acceleration, comfortable deceleration, and rapid deceleration, as shown in the third row in the table 3.2. Using this method to obtain an action sequence for an ego vehicle may result in a significant difference between two adjacent actions, meaning that the jerk value is too high, as we discovered during the development process. In order to get smoother trajectory using the reachable set for path planning, we use the discretization method in the fourth row of the table 3.2. In this method, the interval between actions will have an intersection. Experiments demonstrate that relatively smoother action sequences can be obtained. Assuming that we have obtained the action sequence $(a3, a4)$ by both methods, the acceleration sequence obtained by method 1 is $([-1,\ 1], [1,\ 4])$, while the acceleration sequence obtained by method 2 is $([-2,\ 2], [1,\ 5])$. Evidently, jerk cannot be eliminated from the sampling path using method 1, but method 2 might be able to make jerk disappear.

Table 3.2: Discretization method

Action	a_1	a_2	a_3	a_4	a_5
Behavior	braking	deceleration	constant speed	acceleration	rapid acceleration
Method 1	$[-6, -3]$	$[-3, -1]$	$[-1,\ 1]$	$[1,\ 4]$	$[4,\ 7]$
Method 2	$[-6, -3]$	$[-4, -1]$	$[-2,\ 2]$	$[1,\ 5]$	$[4,\ 7]$

3.3.2 Tree Policy

The tree policy includes selection and expansion. The specific process of implementation is shown in detail in the pseudo-code 2. If the current node is not a terminal node, selection, and expansion are always executed. The definition of the terminal in this project is that the state of the current node is in collision or simulation time for longer than the user-defined simulation time, and neither of the previous two situations happened but the state has reached the preset goal. In general, if a node has no new actions to try, it is called a fully expanded node. Here we refer to the settings in Alpha go [KJ18] and think that the number of visits of the child nodes of the current node is higher than the visited threshold, which is considered to be full expansion. And We set a probability threshold in our program to make expansion not always happen. This design helps the program to trade off explosion and exploitation at a high level, making the algorithm easier to adjust and more robust. In the selection step, select the child node with the largest adapted UCT in formal 2.2. In expansion, if the node N has no action to try, we randomly select from the sub-nodes $\mathcal{N}'_c$ of $\mathcal{N}_c(Node\ N)$ (the set of child nodes of N), whose access times do not exceed the access threshold ρ:

$$\mathcal{N}'_c = \{N_{c,i} | N_{c,i}.visit < \rho, N_{c,i} \in \mathcal{N}_c\} \tag{3.4}$$

According to the distribution of their visit times as equation 3.5.

$$p(N'_{c,i}) = \frac{N_{c,i}.visit}{\sum_{N'_{c,j} \in \mathcal{N}'_c} N'_{c,j}.visit} \tag{3.5}$$

Otherwise, perform normal expansion on a child node with an untried action.

Algorithm 2 Pseudo-code for Tree Policy of SP- MCTS

Input: node
Output: new child node

function TREE POLICE(node)
  while node is non-terminal **do**
    if node is not full-expansion **then**
      % not always expansion, p is a threshold value, set by user
      if $random.uniform(0,1) < p$ **then**
        $node \leftarrow selection(node)$
      else
        $newchildnode \leftarrow expansion(node)$
        return $newchildnode$
      end if
    else
      $node \leftarrow selection(node)$
    end if
  end while
  return $node$
end function

function SELECTION(node)
  $v_i, n_i \leftarrow node.state$
  $\bar{X} \leftarrow \frac{v_i}{n_i}$
  return $argmax_{i \in I} \left(\bar{X} + C \times \sqrt{\frac{Inn_p}{n_i}} + \sqrt{\frac{\sum x^2 - n_i \cdot \bar{X}^2 + D}{n_i}} \right)$
end function

function EXPANSION(node)
  if node has no untried action **then**
    filter out child node with $n_i < \rho(visitthreshold)$
    choose one *childnode* based on visit distribution
    return *childnode*
  else
    choose $a \in$ untried action
    add *childnode* with action a to *node*
    update state s' of *newchild* with $f(node.s, a)$
  end if
  return *childnode*
end function

3.3.3 Default Policy

In the default policy stage, we will introduce in detail how to complete the interaction between vehicles and how to calculate the reward for the node. In the pseudo-code 3, we have outlined the simulation process.

Algorithm 3 Pseudo-code for Default Policy of SP- MCTS

Input: node
Output: reward

```
1: function DEFAULT POLICE(node)
2:     sample K paths within action a in node
3:     for each path in sampled paths do
4:         classify vehicles
5:         create simulation scenario
6:         update scenario
7:         evaluate the path
8:         while path is non-terminal do
9:             update ego vehicle state with heuristic randomly selected action
10:            update other vehicle with IDM behavior model
11:            evaluate the path
12:        end while
13:        calculate evaluate cost and record simulation results (collision, reach goal position)
14:    end for
15:    reward ← statistically record results, evaluate action a
16:    return reward
17: end function
```

Vehicle Classification and Sequence Decision

In order to complete the interactive simulation, we adopt a scheme similar to that in the paper [LKK16]; the ego vehicle is controlled by the action system, while the other vehicles are controlled by the IDM vehicle behavior model. The ego vehicle does not need to communicate with other vehicles to complete cooperative motion planning. This approach is more computationally effective than cooperative motion planning based on joint action in the multi-agent system. Since the performance of MCTS is mainly determined by its effective search depth, the multi-agent problem has inherent large branch factors leading to the curse of dimensionality [BPW+12].
We construct a sequential decision chain by classifying the vehicles in the scenario and evaluating their priority. As shown in figure 3.8, the vehicles in the environment are divided into five categories: ego vehicle, direct influences vehicles, influence vehicles, indirect influence vehicles, and non-influence vehicles. The definition for these five categories are listed below:

- ego vehicle: The ego vehicle is the vehicle for which we need to plan. Direct influences vehicle and influence vehicles have an impact on behavior of ego vehicle.

- direct influences vehicle:The Euclidean distance between the direct influences vehicle and the ego vehicle is less than the relative safety distance. They are both in the same lane, direct influences vehicle in front of the ego vehicle.
- influence vehicles: A vehicle is an influences vehicle when the Euclidean geometric distance between the vehicle and the ego vehicle is smaller than its relative safe distance.
- indirect influence vehicles: When a vehicle is considered to be an indirect influences vehicle, it must be within a relatively safe distance from the direct influences vehicle. Where a vehicle meets the criteria for influence vehicles and indirect influence vehicles, it is classified in the category of indirect influence vehicles.
- non-influence vehicles: The group of vehicles that do not match the aforementioned requirements is known as the non-influence vehicles.

For this a relative safety distance $d_{r,s}$ (in m) will be defined:

$$d_{r,s} = \frac{v_c \times 3.6}{2} \tag{3.6}$$

v_c is the speed of the observed vehicle, and the unit is m/s. This idea comes from the German driving rules. On the highway, it is necessary to maintain a relative safe distance from the vehicle in front. The relatively safe distance is half of the speed indicator(km/h) and is an empirical value. For example, if the vehicle is driving at 80 km/h, then the relative safe distance to be kept should be 40 m.

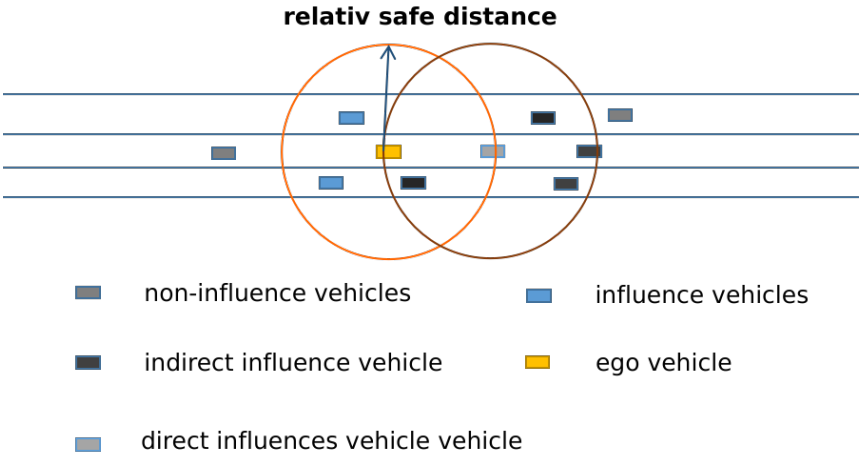


Figure 3.8: Classification of vehicles based on relative safe distance

After the vehicle is classified, the decision chain of each type of vehicle is shown in figure 3.9. The behavior of indirect influence vehicles is first determined and non-influence vehicles will be removed from the simulated scenario. Then the direct influences vehicle uses the IDM model to respond to the behavior of indirect influence vehicles. Subsequently, the ego vehicle uses the

action system control to react to the influences vehicle. Finally, the influence vehicles react to the ego vehicle. We classify the vehicles more precisely than was done in the paper [LKK16] to decrease the number of vehicles that must be taken into account during the simulation. This can save a lot of time, especially in scenarios with a lot of vehicles. The difficulty with our approach is to identify behaviors of direct influences vehicle, whose behavior prediction accuracy directly affects the performance of the algorithm. For this reason, we limit the simulation time to a certain length, and if the vehicle reaches the predetermined goal within the limited simulation time, the simulation is terminated early. If the simulation time is too long, the simulation results will lose credibility and have a huge deviation from the actual situation. During simulation time, it is assumed that the behavior of indirect influence vehicles will not change significantly, and continue to maintain the speed at the moment of the simulation. Although this is inaccurate, it provides some error tolerance because the action is in a continuous range. Subsequent experimental results also prove that this method is effective.

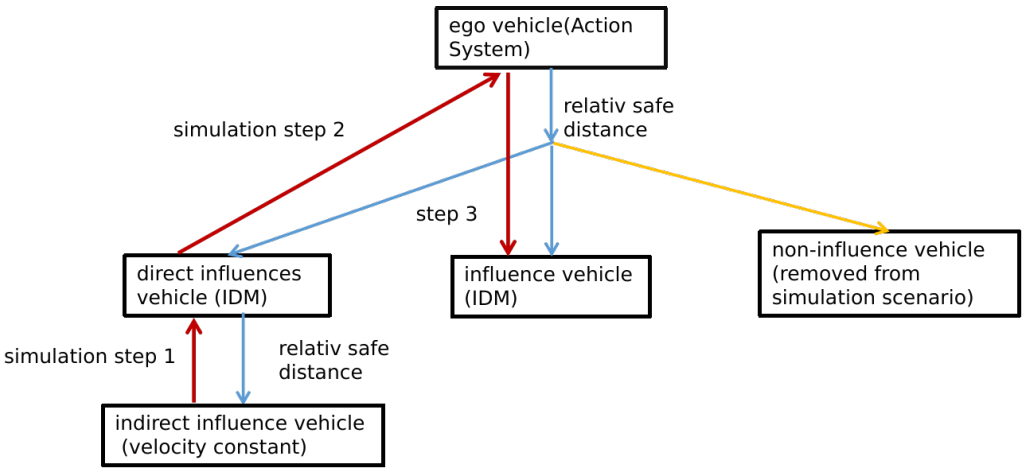


Figure 3.9: Decision chain for classified vehicles

Sampling Path and Evaluation

We employ a sampling method to assess the continuous action space. The criterion for its evaluation mainly includes collision rate and reaching rate. As shown in figure 3.10, the ego vehicle will use one action to travel for 5 sampling times, and the blue-gray block shows the position that the vehicle can reach at each sampling time. The red path on the left is obtained using the minimum acceleration, and the blue path on the right is obtained using the maximum acceleration. using the sampled acceleration, the path in the middle is created.
At the beginning of the simulation, the root node contains only one state pair (v_0, p_0). As the node executes the action, the generated node contains three sate pairs $\{(v_{min}, p_{min}), (v_{mid}, p_{mid}), (v_{max}, p_{max})\}$, corresponding to acc_{min}, acc_{mid}, acc_{max} respectively. A continuous state space can be generated from a single state pair. For example, if action a_2 is executed from $(v_{min,1}, p_{min,1})$, the generated set is $\{(v_{min,2}, p_{min,2}), ...(v_2, p_2)\}$, that is, the area of the triangle between the blue arrows and the red arrow in the figure 3.11. If we push back from state (v_2, p_2) to time

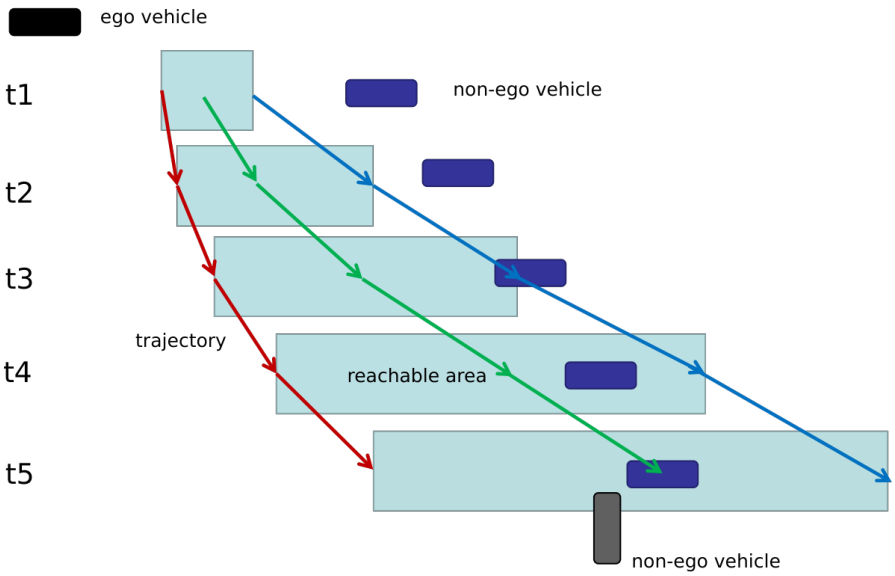


Figure 3.10: Drivable area at five sample times

step t_2, we can find countless solutions, which is not good for sampling. When we sample v from $[v_{min,1}, v_{max,1}]$ and p from $[p_{min,1}, p_{max,1}]$, it is quite possible to get a state pair that does not exist, like $(v_{min,1}, p_{max,1})$. It is difficult to verify whether the state pair is reasonable. One of the solutions is to compute the reachable set and then obtain some paths by sampling in the reachable set. These paths are guaranteed to be dynamically feasible and collision-free. But this approach is difficult to implement, and computing the reachable set in simulation is expensive. So here we use a simpler way to generate sampled trajectory based on the state pair that are sorted in the node. Through the equation 3.7 we can get the sampled state pair $(v_{t+1,s}, p_{t+1,s})$, where acc_s is randomly sampled from the acceleration interval.

$$\begin{aligned} v_{t+1,s} &= v_{t,x} + acc_s t \\ p_{t+1,s} &= p_{t,x} + v_{t,x} t + \frac{1}{2} acc_s t^2 \\ (v_{t,x}, p_{t,x}) &\in \{(v_{t,min}, p_{t,min}), (v_{t,mid}, p_{t,mid}), (v_{t,max}, p_{t,max})\}, \\ acc_s &\in \{acc \mid acc_{t+1,min} \leq acc \leq acc_{t+1,max}\} \end{aligned} \tag{3.7}$$

After generating the sampled paths, each path will be used for interactive simulation with other vehicles and evaluated. For the ego vehicle, the evaluation cost includes the following items:

- j_r: Cost for difference between simulated and reference trajectories (generated by CommonRoad Route Planner API), $\sum \|p_i - p_{r,i}\|$
- j_v: Cost different from expected velocity, $\|v_i - v_d\|$
- j_d: Cost for distance to other vehicles, $\sum \frac{1}{d_i}$

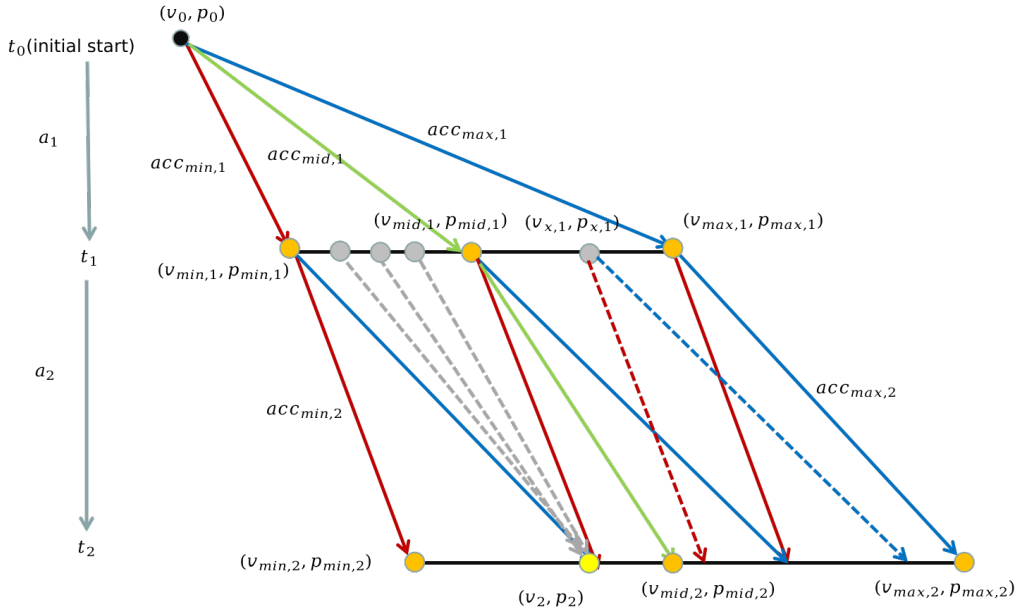


Figure 3.11: Sampling path

- j_g: The distance cost to the destination, if the ego vehicle does not reach the destination location, $\|p_i - p_d\|$
- j_c: Cost for collision, if a collision occurs. This value is very large.
- j_t: Cost for time, $(t_N - T_s)t_N$

The total of all of these cost terms is:

$$J_{ego} = \sum w_{i,.} J_{i,.} \tag{3.8}$$

with weight parameters $w_{i,.}$ for each cost term.
After the evaluation cost has been calculated for the K paths sampled, we perform statistics on the information of the K paths to evaluate the entire action. There are collision rate β_c and reach rate β_c as evaluation criteria used. If there are K_1 paths in the K paths that collide and K_2 paths reach the goal position. Then the rate can be obtained by the following equation.

$$\beta_c = \frac{K_1}{K} \tag{3.9}$$

$$\beta_r = \frac{K_2}{K} \tag{3.10}$$

We can set thresholds ρ_c and ρ_r for these two rates respectively, determined through experiments. If the value of the collision rate is greater than the threshold ρ_c, it is considered that using the current action will cause the vehicle to enter the state of collision. For reach rate $(\beta_r > \rho_r)$, it means that the vehicle has reached the goal position.

Now we can define a terminal node. As long as the node meets one of the three conditions, the node is terminal: (1) The simulation time of the current node is greater than the set plan horizon time length. (2) The state of the node is in the collision. (3) The node has reached the goal position.

Heuristic Random Selection

During the simulation of this project, the ego vehicle updates the state by heuristic randomly selecting action. The behavior of the vehicle is based on road information and information from other vehicles. The road information is the specified desired velocity v_d. Assuming that there is only an ego vehicle on the road, the behavior of the vehicle only refers to the desired velocity. For example, if the speed of the vehicle v_e is lower than the expected speed, the priority of the vehicle acceleration behavior should be higher than that of constant speed and deceleration, and the probability of being selected is higher, such as $0.5v_d < v_e < 0.9v_d$, comfortable acceleration has priority over rapid acceleration. The table 3.3 shows the setting used in this project.

Table 3.3: Probability distribution for choosing an action is based on the speed of vehicle v_e and desired speed v_d

	$v_e > 1.5v_d$	$1.1 \leq v_e \leq 1.5v_d$	$0.9v_d \leq v_e < 1.1v_d$	$0.5v_d \leq v_e < 0.9v_d$	$v_e \leq 0.5v_d$
a1	0.3	0.25	0.1	0.15	0.15
a2	0.25	0.3	0.25	0.15	0.15
a3	0.15	0.15	0.3	0.15	0.15
a4	0.15	0.15	0.25	0.3	0.25
a5	0.15	0.15	0.1	0.25	0.3

The reference to the information of other vehicles only refers to the direct influences vehicle, and the two parameters of TTC (time to collision) and TIV (inter-vehicular time) are used for risk assessment of ego vehicle behavior. This method is proposed in the article [GVM+10]. TTC is the time it would take for two cars to collide if they continued on the same lane at the current speed.

$$TTC = \frac{D_{ef}}{v_e - v_f} \tag{3.11}$$

D_{ef} is the distance between the ego vehicle and the direct influences vehicle. v_e is the considered target speed of the ego vehicle and given speed v_f of the direct influences vehicle. TIV stands for the time to collide with the front vehicle at the current speed assuming that the vehicle in front does not move, which is used to help TTV assess the risk.

$$TIV = \frac{D_{ef}}{v_e} \tag{3.12}$$

P_{TTC} and P_{TIV} are the possibility of collision associated with TTC and TIV respectively, as shown in figure 3.12. From this, we calculate the final risk by

$$R = P_{TTC}(v_e)(v_f - v_e) + P_{TIV}(v_e)max(v_f - v_e, v_f - \gamma v_f TIV - v_e) \tag{3.13}$$

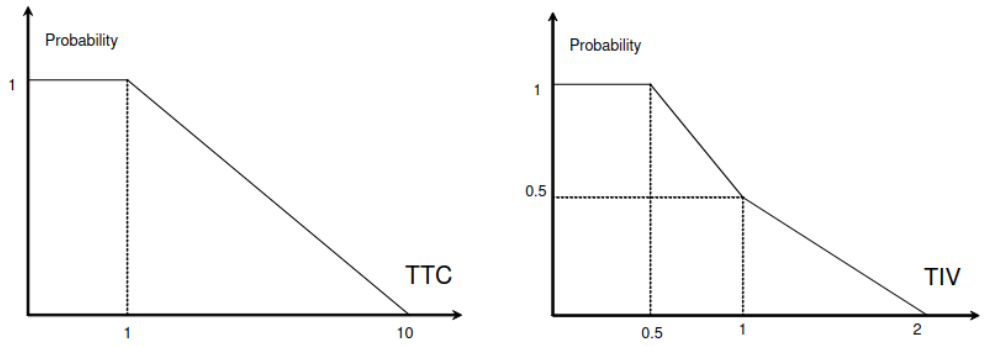


Figure 3.12: TTC (left) and TIV (right) based possibility of collision(cited from [GVM+10])

Since each action is a continuous set, we will sample the acceleration and calculate the expectation of risk. The obtained risk expectation for each action will be sorted. The higher the risk expectation, the lower the probability of the action being selected. For example, if the obtained risk expectation list is $[1, 2, 3, 4, 5]$, then the order of the probability of the action being selected is $[5, 4, 3, 2, 1]$, and convert it into probability $[\frac{5}{15}, \frac{4}{15}, \frac{3}{15}, \frac{2}{15}, \frac{1}{15}]$. Finally, the two probabilities are added and averaged to obtain the action probability distribution of the ego vehicle. If $v_e > 1.5v_d$, the choose probability based on road information is $[0.3, 0.25, 0.15, 0.15, 0.15]$, than the finally choose probability is $[0.32, 0.258, 0.175, 0.142, 0.108]$.

3.3.4 Interaction-Aware Reachable Set

After completing the MCTS simulation, an action list can be obtained from the final node. We count how many time steps each action is executed. And we use the method implemented by commonRoad-Reach API to calculate the reachable set. What needs to be reminded is that in the specific implementation process: (1) Zero state polygon in longitudinal direction needs to be recalculated every time the action is updated. (2) There is no need to prune those that cannot reach the end node after calculating the reachable set with a specific amount of time step after each calculation, but they should be pruned after finishing the entire time. The display of specific simulation results will be shown in the section 4.

Chapter 4

Evaluation

This section will mainly show the performance of the algorithm in various scenarios, and compare the calculated interaction-aware reachable set with the original calculated reachable set. In this project, all of the test scenarios were taken from CommonRoad. The scenario used must already have a target position defined for the ego vehicle, since we need to establish a reference trajectory before performing the SP-MCTS simulation. The table 3.1 in chapter 3.2.2 contains the ego vehicle's specifications. The SP-MCTS parameters are configured as indicated in the table 4.1.

Table 4.1: Parameter setting of SP-MCTS in the experiment

Parameter	Description	Value
T_s	Planning horizon time	$3s$
$\mathcal{A}$	Discretized action set with 5 actions	$\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5\}$
n_s	Number of simulations per root node (time budget)	10
ρ	Visit time threshold	4
C	Constant value in UCT of SP-MCTS	0.3
D	Constant value in UCT of SP-MCTS	10000

4.1 One Lane Scenario

Shown in the figure 4.1 is the comparison of simulation results using scenario DEU_IV21-1_1_T-1 at 1s, 2s, and 3s respectively. The red dot is the initial state of the ego vehicle, its initial speed is 12 m/s, the target location is the end of the current lane and the desired velocity is 15 m/s. The blue rectangle represents the surrounding vehicle, which maintains the current lane and drives at a constant speed of 10 m/s. The gray blocks are the reachable set of the ego vehicle. The three sub-pictures on the left in the figure represent the reachable set calculated using $([-6, 7], 30)$ (30-time steps with acceleration interval $[-6, 7]$), and the three sub-pictures on the right are the interaction-aware reachable set calculated based on SP-MCTS. The interaction-aware reachable set differs from the original reachable set in that it is more refined. The original reachable set at 3s includes two potential strategies following the leader vehicle or changing

lanes to overtake.If these reachable sets are utilized to determine the trajectory, definitely we need to consider which strategy to choose based on some risk assessments. This will make the calculation more expensive and decrease the benefit of using reachable sets. The strategy shown in the interaction-aware reachable set is to follow the leader vehicle, and the interaction-aware reachable set can be applied straight away for trajectory planning.

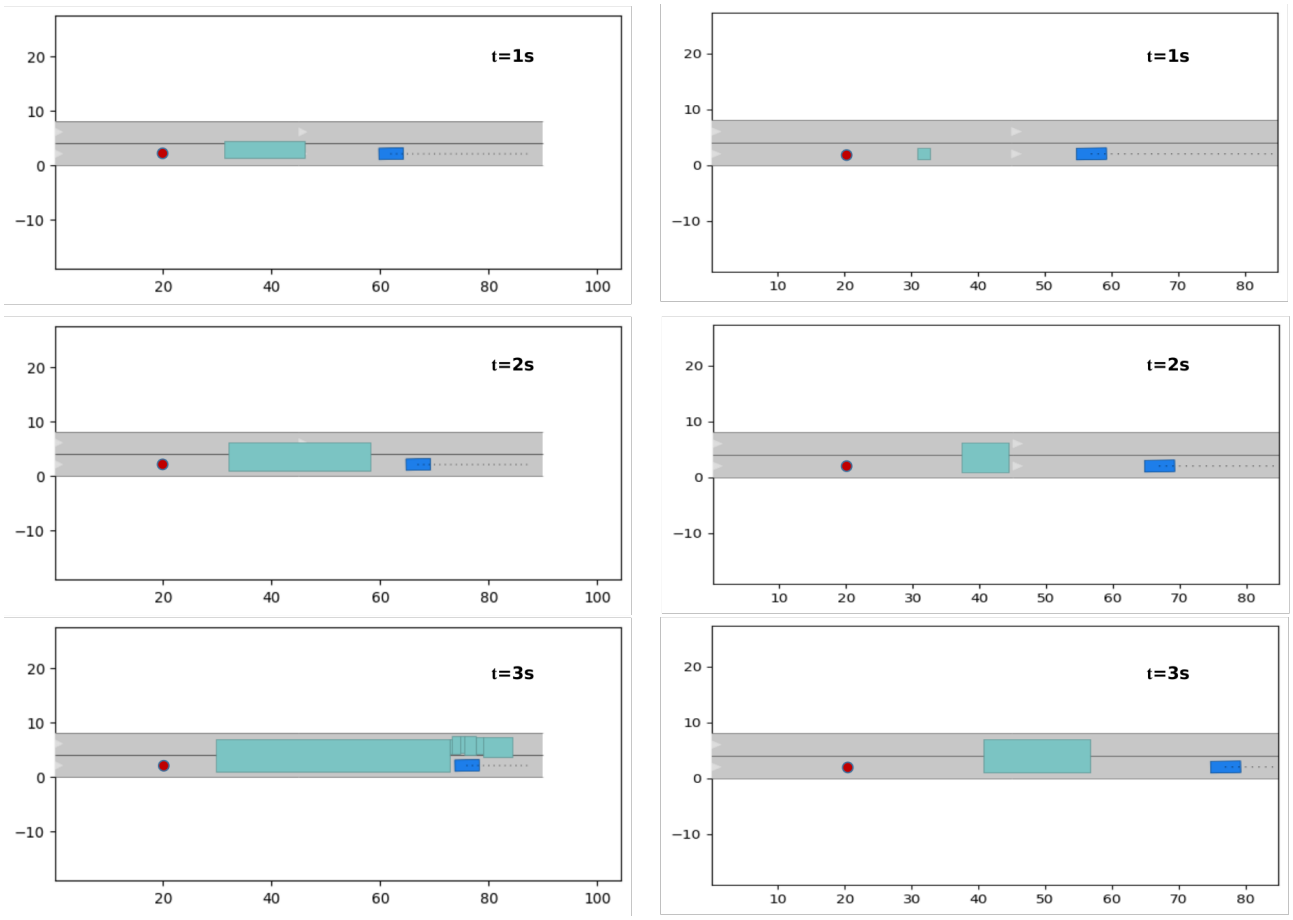


Figure 4.1: Simulation result in scenario DEU_IV21-1_1_T-1 at $1s$, $2s$ and $3s$, the three on the left are original reachable set with $([-6, 7], 30)$, and the three on the right are interaction-aware reachable set with $[([-2, 2], 5), ([-4, -1], 5), ([-6, -3], 5), ([-2, 2], 10), ([-6, -3], 5)]$

The comparison of simulation results employing scenario DEU_Moelln-7_1_T-1 at 2s and 3s is presented in the figure 4.1. The initial velocity is 11 m/s of the ego vehicle, the target location is the end of the current lane and the desired velocity is 15 m/s. The non-ego vehicle keeps the current lane with an initial velocity of 9 m/s and continues to accelerate with an average acceleration of 1.9 m/s^2. The purpose of this scenario is to evaluate how well the algorithm works on the curve. The interaction-aware reachable set remains behind the leader vehicle, which represents the following maneuver. According to interaction-aware reachable set results at 3 s, certain reachable areas bypass the leader vehicle from adjacent lanes. This occurs as a result of the fact that our actions are continuous and we evaluate the entire through

sampling. For instance, if the optimal action interval is $[-2, 0]$, but we can only choose $[-2, 2]$, the computed reachable set with action will be bigger than the ideal. Second, because the reachable set propagation exhibits inheritance properties, the reachable set will carry over any errors from earlier reachable set calculation. The driving direction of the adjacent lane differs from the driving direction of the lane where the ego vehicle is located, making this portion of the reachable set unrealistic. This is due to the reachable set computation not taking into account traffic rules.

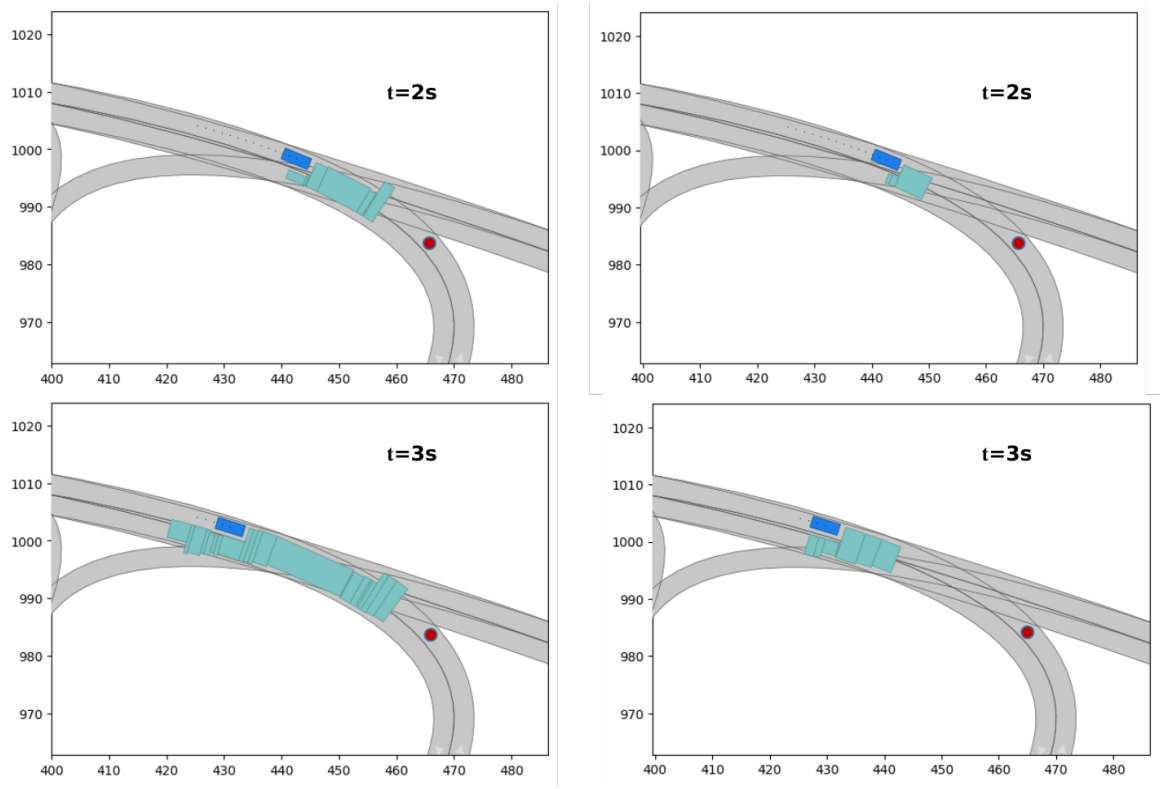


Figure 4.2: Simulation result in scenario DEU_Moelln-7_1_T-1 at $2s$ and $3s$, the two on the left are original reachable set with $([-6, 7], 30)$, and the two on the right are interaction-aware reachable set with $[([-2, 2], 5), ([-4, -1], 5), ([1, 5], 5), ([4, 7], 5), ([-6, -3], 5)]$

The comparison of the original reachable set at 2.1s and 2.7s is shown in figure 4.3. We can notice that the reachable set has a yellow area for a backwards drive throughout the continuous propagation phase. Vehicles drive in the opposite direction to the road in a reverse zone. Because the propagation process fixes acceleration bounding without accounting for the velocity constraint of the ego vehicle. There won't be a retreat area in the interaction-aware case because velocity constraint will be taken into consideration during calculating the interaction-aware reachable set.

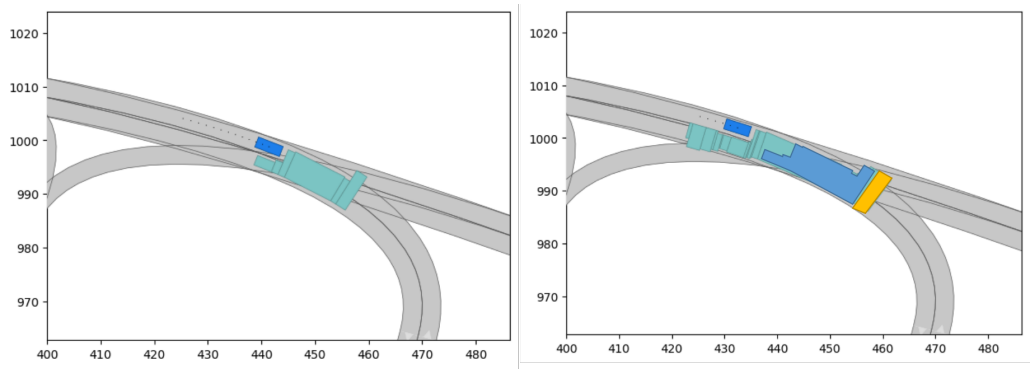


Figure 4.3: Original reachable set with scenario DEU_Moelln-7_1_T-1 at 2.1s(left) and 2.7s(right). The reachable set of 2.1s shown in blue, while the area to retreat is shown in yellow.

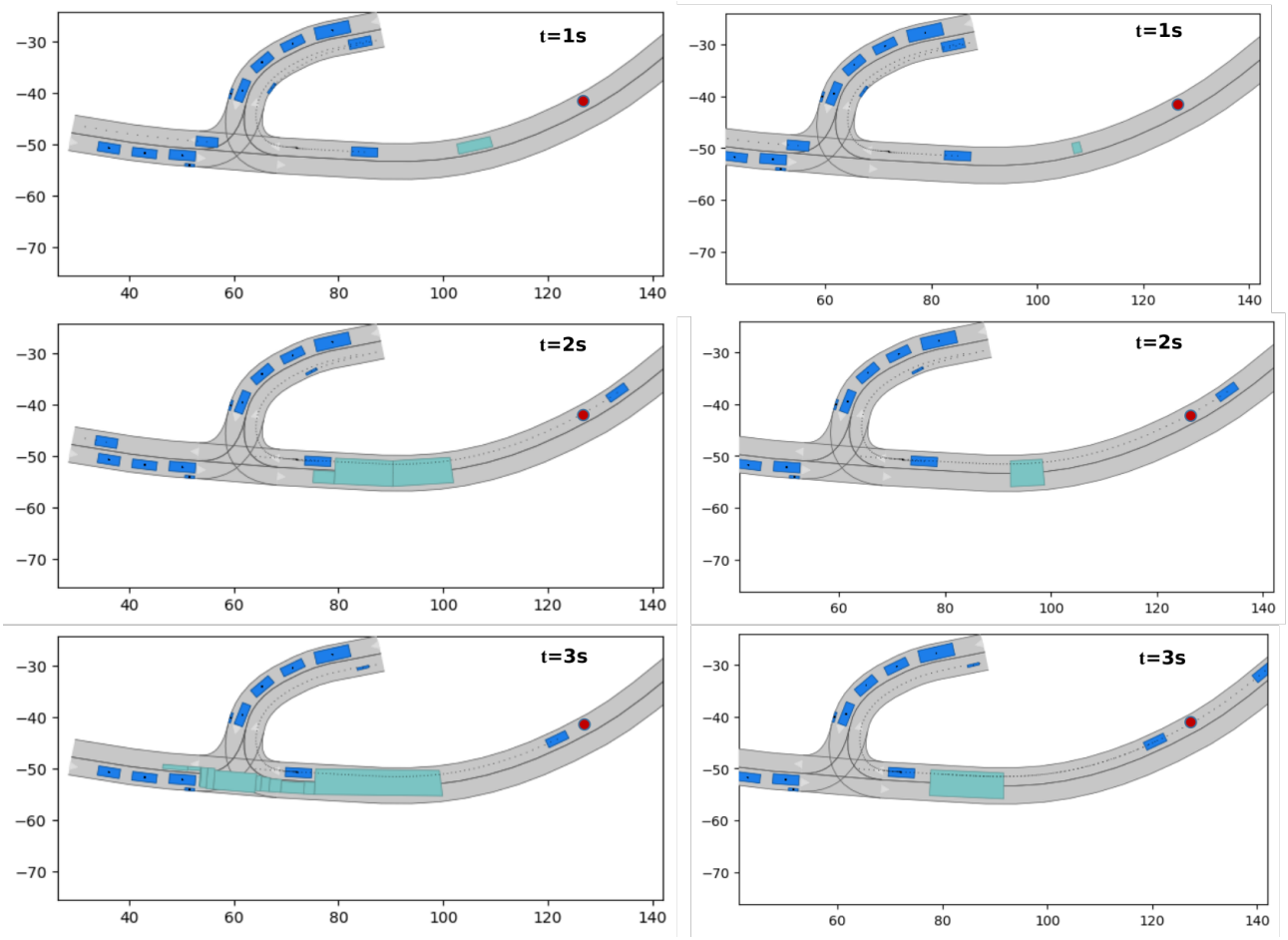


Figure 4.4: Simulation result in scenario DEU_Lohmar-32_1_T-1 at 1s, 2s and 3s, the three on the left are original reachable set with $([-6, 7], 30)$, and the three on the right are interaction-aware reachable set with $[([-4, -1], 10), ([-6, -3], 10), ([-2, 2], 5), ([4, 7], 5)]$

The simulation scenario used in figure 4.4 is DEU_Lohmar-32_1_T-1. In this scenario, the leader vehicle of the ego vehicle brakes quickly to stop. This leader vehicle will be identified as a direct influence vehicle in the SP-MCTS simulation stage. Its behavior simulation will employ IDM, with the desired speed of the vehicle set at 15 m/s. This is very different from the actual situation, which will cause the simulation to be inaccurate or perhaps fail. As shown in figure 4.5, the expected speed of the leader vehicle is set to 15 m/s and simulated for 4 seconds to obtain an interaction-aware reachable set. The action list generated by SP-MCTS exhibits a strategy of uniform speed or acceleration as a whole, rather than the optimum deceleration. The interaction-aware reachable sets at 4s completely exceed the leader vehicle. Therefore, this simulation is invalid. To ensure that the interaction-aware reachable set is always kept behind the leader vehicle, as depicted in the figure 4.4, we artificially set the desired speed of the leader vehicle to $0m/s$. The above analysis shows that the performance of the algorithm is related to whether the surrounding vehicle behavior can be accurately simulated.

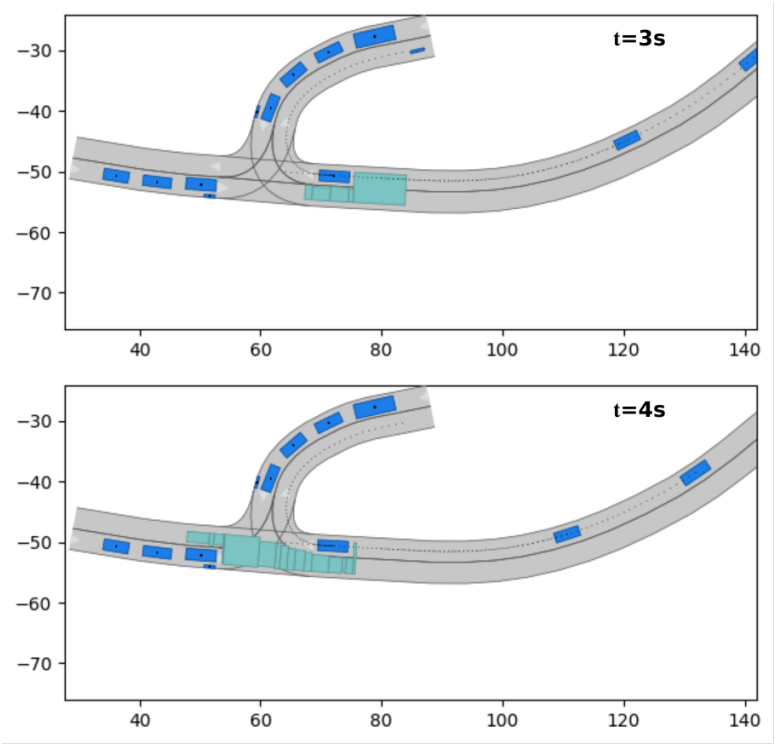


Figure 4.5: Interaction-aware reachable set in scenario DEU_Lohmar-32_1_T-1 at $3s$ and $4s$ with $[([-2,2],10),([-4,-1],10),([-2,2],5),([1,5],5),([-2,2],5),([-6,-3],5)]$

In the case study DEU_Lohmar-13_1_T-1 in figure 4.6, the ego vehicle is chased by a vehicle from behind and there are no vehicles in front of the ego vehicle. The interaction-aware reachable set demonstrates that the ego vehicle will always maintain a certain distance from the rear vehicle and avoid making contact. The reverse driving portion has been removed from the reachable set compared to the origin.

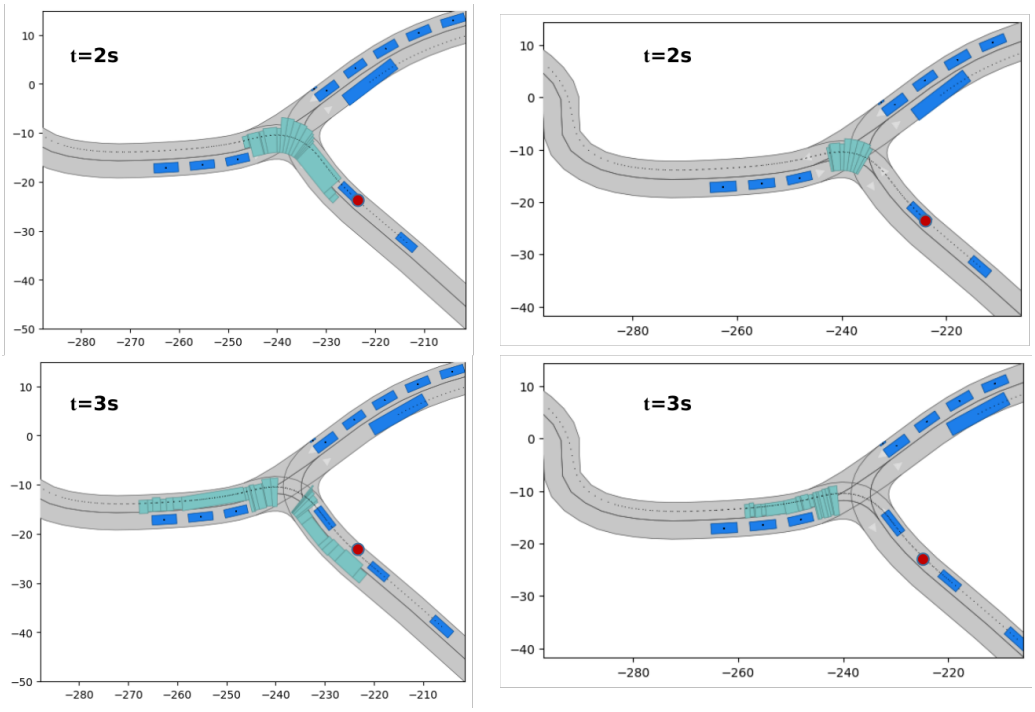


Figure 4.6: Simulation result in scenario DEU_Lohmar-13_1_T-1 at 2s and 3s, the two on the left are original reachable set with $([-6, 7], 30)$, and the two on the right are interaction-aware reachable set with [([1, 5],15), ([-2, 2],10), ([4, 7],5)]

4.2 Intersection Scenario

This project mainly discretizes longitudinal acceleration. The discrete acceleration interval yields the matching part of reachable sets. The interaction-aware reachable sets should eventually be able to demonstrate macro-level driving strategies. The intersection scenario is an efficient way to evaluate it. At the intersection, whether the ego vehicle needs to speed up or slow down to wait for other vehicles to pass through is the most important decision. So this section uses the scenarios including the intersection for simulation.

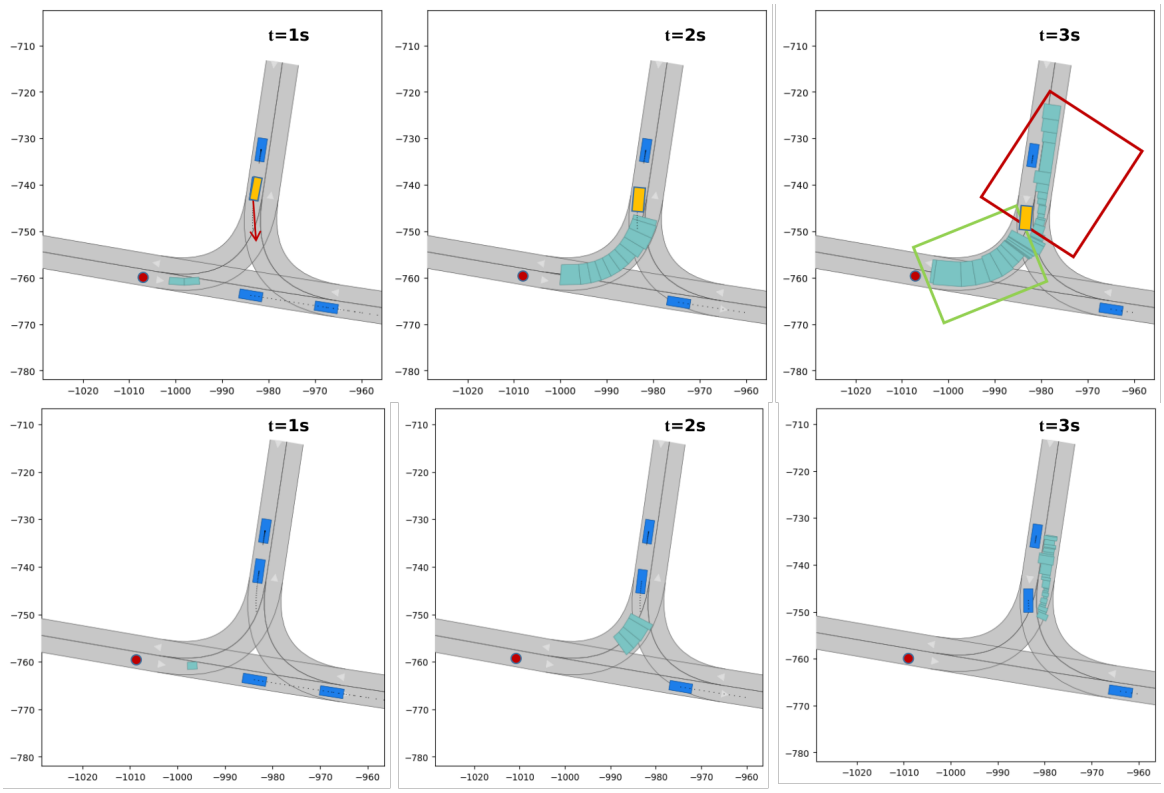


Figure 4.7: Simulation result in scenario ESP_Almansa-6_1_T-1 at 1s, 2s and 3s, the three on the top are original reachable sets with $([-6, 7], 30)$, and the three on the bottom are interaction-aware reachable sets with [([1, 5],15), ([-2, 2], 5), ([-4, -1], 5), ([-6, -3],5)]

The simulation results in figure 4.7 use the scenario ESP_Almansa-6_1_T-1. The lower three sub-pictures represent the interaction-aware reachable set, whereas the upper three sub-pictures are the original reachable set. When the ego vehicle passes through the intersection in this scenario, it is necessary to consider the situation of vehicles coming from the left. During the simulation, the expected speed of the left vehicle (orange vehicle) will be set according to its motion, instead of simply being set to 15 m/s. The expected speed of the ego vehicle is still 15 m/s. The original reachable set at 3s is divided into two possibilities by the left vehicle, one (yellow frame) is to wait for the left vehicle to pass, and the other (red frame) is to pass before the left vehicle. The interaction-aware reachable sets at 3s show that it is passing before the vehicle on the left.

Figure 4.8 is the result of using scenario DEU_Lohmar-20_1_T-1 simulation. In the beginning, the directly affected vehicle of the ego vehicle is the yellow vehicle in figure 4.8, which accelerates through the intersection. After the vehicle moved away, the direct influences vehicle of the ego vehicle turned into a red vehicle, driving towards the intersection from the left direction and slowly accelerating through the intersection. If the vehicle accelerates to follow the yellow vehicle through the intersection, it will collide with the red vehicle, just like the original reachable sets at 2s and 3s. The more reasonable driving behavior of the ego vehicle should

be to slow down and stop, and then let the red vehicle pass the intersection first. The policy expressed by the interaction-aware reachable set is consistent with the ideal maneuver.

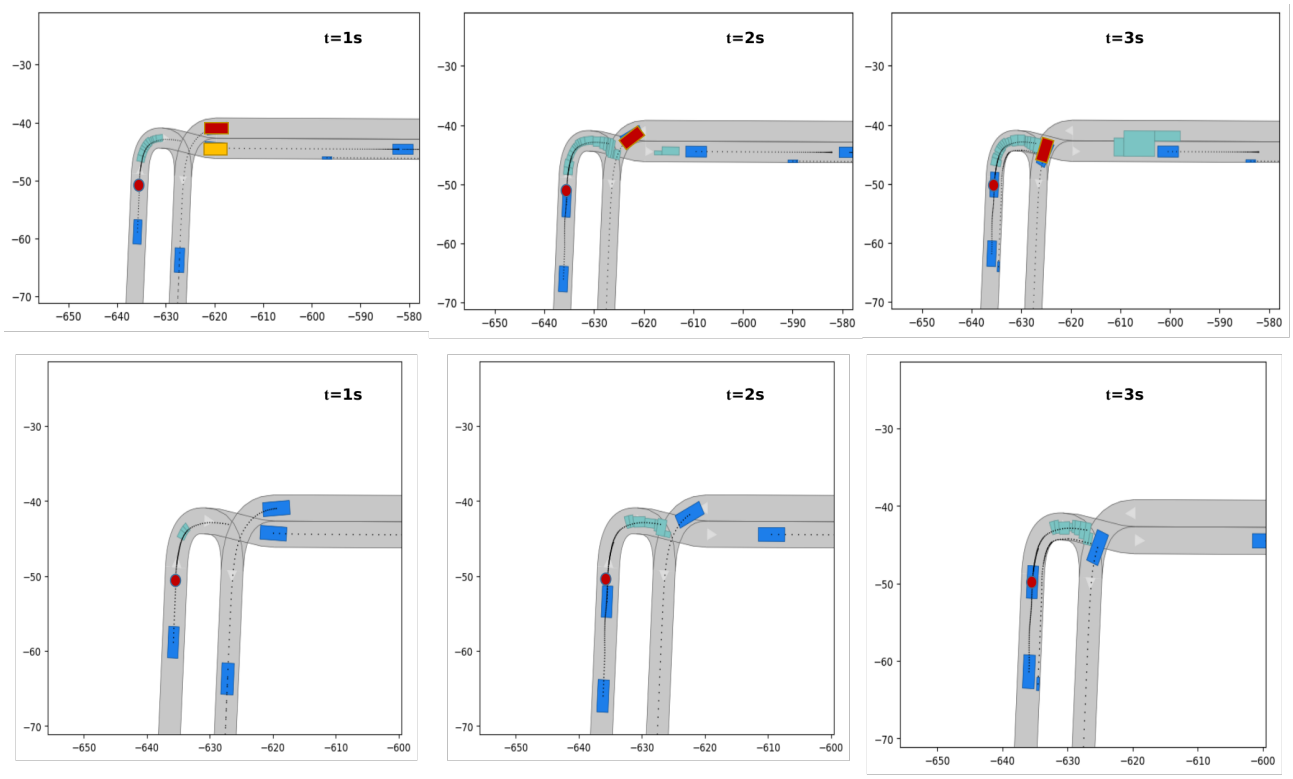


Figure 4.8: Simulation result in scenario DEU_Lohmar-20_1_T-1 at 1s, 2s and 3s, the three on the left are original reachable set with $([-6, 7], 30)$, and the three on the right are interaction-aware reachable set with [([-2, 2],5), ([-4, -1],5), ([-2, 2],5), ([-6, -3],1), ([-2, 2],1), ([-1.94, 2],1), ([0.0, 2],8), ([1, 5],5)]

This scenario ZAM_Intersection-1_2_T-1 is a typical intersection. The surrounding vehicle on the right passes through the intersection at a constant speed of 16 m/s. Through the previous analysis, we know that the ego vehicle at the intersection theoretically has two macro driving strategies. In the previous two scenarios, we can determine one driving strategy by analyzing the state of each traffic participant. And it will be compared with the strategy represented by the interaction-aware reachable set. However, both driving strategies are possible in this scenario (see figure 4.9), which can be used to test whether the algorithm can give a relatively certain maneuvering strategy. If we set the desired speed of the ego vehicle to be 15 m/s, then the ego vehicle will pass through the intersection earlier than the vehicle on the right, and the simulation results can be seen in figure 4.10. And when we set the expected speed of the ego vehicle to $6m/s$, the vehicle will slow down and wait for the vehicle on the right to pass the intersection and then accelerate to pass. The simulation results can be viewed in figure 4.11. Experiments prove that the algorithm can find corresponding better driving strategies under different desired velocity conditions.

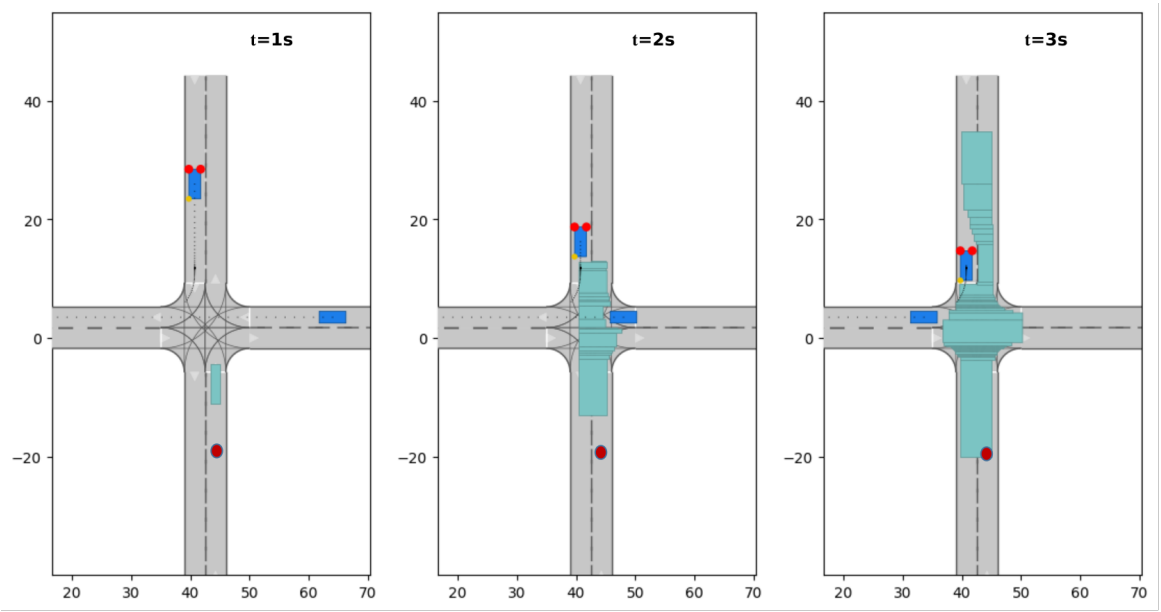


Figure 4.9: Original reachable sets in scenario ZAM_Intersection-1_2_T-1 at 1s, 2s and 3s with $([-6, 7], 30)$

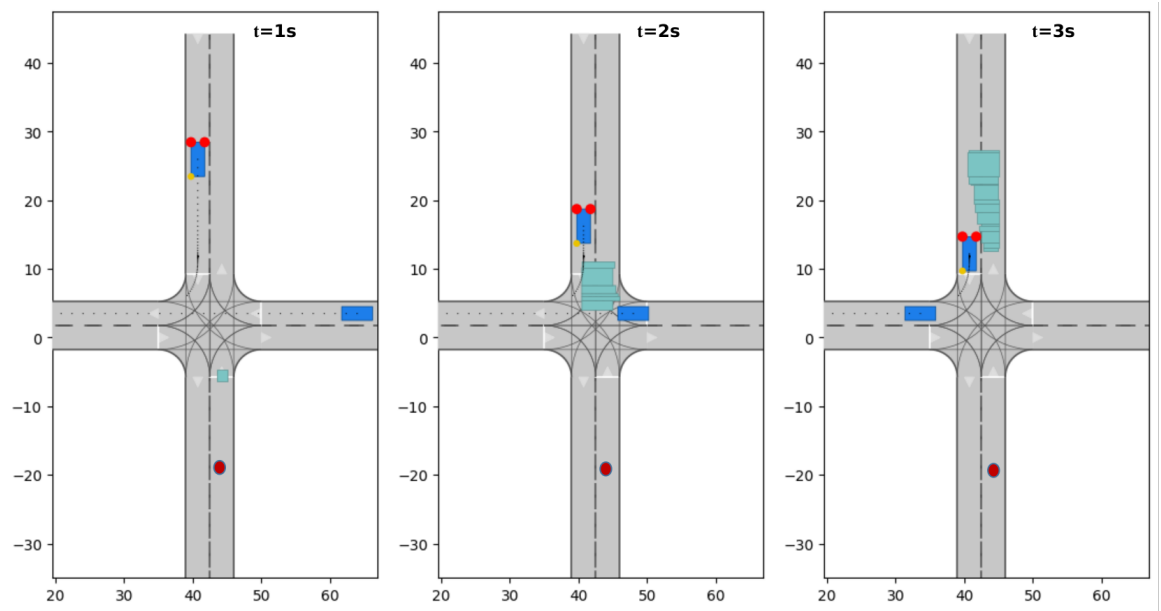


Figure 4.10: Interaction-aware reachable sets in scenario ZAM_Intersection-1_2_T-1 at 1s, 2s and 3s with $[([-2, 2], 5), ([-4, -1], 5), ([-6, -3], 5), ([-2, 2], 4), ([-1.9, 2], 1), ([1, 5], 5), ([-0.9, 2], 1), ([0, 2], 4)]$, the desired velocity $v_d = 6m/s$

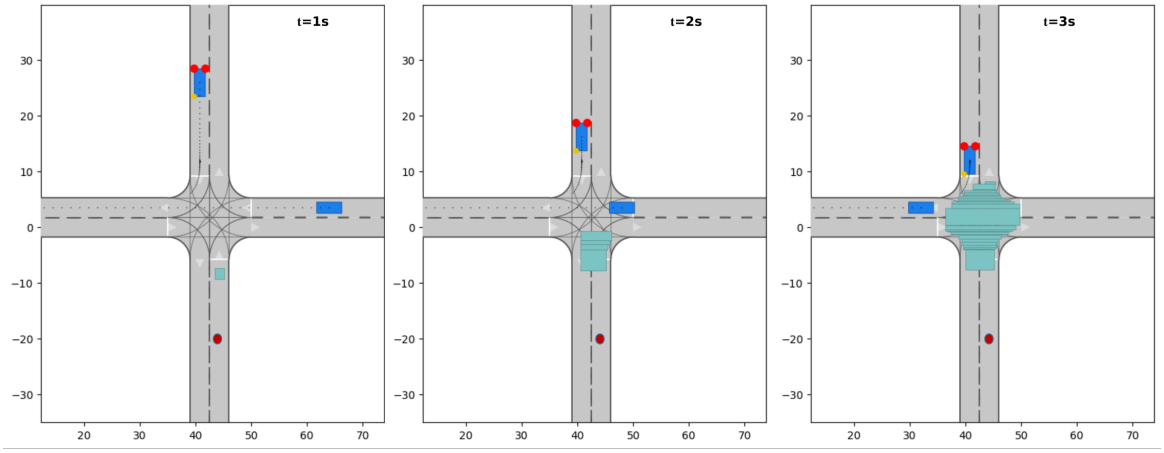


Figure 4.11: Interaction-aware reachable sets in scenario ZAM_Intersection-1_2_T-1 at 1s, 2s and 3s with $[([4,7],5),([1,5],10),([4,7],5),([-6,-3],5),([-2,2],5)]$, the desired velocity $v_d = 15m/s$

Chapter 5

Discussion

In chapter 4, we used 4 one lane scenarios and 3 intersection scenarios to conduct experiments. In each of the test scenarios evaluated, interaction-aware reachable sets searched by SP-MCTS clearly demonstrate suitable driving strategies in the longitude direction. In contrast to the original reachable sets, interaction-aware reachable sets eliminate the backward behavior of the ego vehicle.

Through experimentation, we also identified two aspects where this algorithm could be improved. First, the algorithm relies on the prediction of the behavior of the surrounding vehicles, and the vehicle behavior model IDM used in the algorithm cannot simulate the behavior of surrounding vehicles well in some scenarios. It is necessary to manually adjust the IDM parameters for the surrounding vehicles to obtain reasonable simulation results. For example, in scenario s DEU_Lohmar-32_1_T-1 in figure 4.4, the vehicle in front of the ego vehicle brakes suddenly, but there are no vehicles in front of this leader vehicle. According to the sequential decision introduced by the tree policy, the leader vehicle will be controlled through the IDM behavior model during the simulation phase. The desired speed of all vehicles will be set to be the same 15 m/S during the simulation phase. Then the leader vehicle will accelerate to the expected speed with acceleration $\dot{v}_{\alpha}^{\text{free}}$ (equation 2.5), which is inconsistent with the actual situation. The obtained simulation results are also invalid. when we manually set the desired speed of the leader vehicle to 0 m/s, valid results can be obtained. There are many situations where manual settings are required at intersection scenarios because the IDM behavior model (car following model) cannot well simulate the behavior of vehicles at intersections. A possible solution is to dynamically set the desired speed according to the historical information of the vehicle. For example, if the historical information of the vehicle is deceleration, then we consider that the expected speed of the vehicle should be lower than the current speed. Alternatively, choose a different vehicle behavior model that performs better, albeit this is challenging. Second, we evaluate actions through sampling, so there will be evaluation errors. The calculation of the reachable set at time t_k needs to be based on the reachable set at time t_{k-1}, so there will be cumulative errors. These errors are acceptable in the final simulation results, such as in case DEU_Moelln-7_ 1_T-1, but it is not efficient. A possible optimization method is to refine the acceleration interval, that is, to increase the number of actions. Since one action corresponds to one acceleration interval, we can modify the interval boundary in

SP-MCTS per the circumstances of each node to achieve better discretization. This is another possibility.

Chapter 6

Conclusion

This project successfully realized the use of single-player Monte Carlo tree searches to calculate the interaction-aware reachable sets of automated vehicles. The algorithm verified the usability of single-player Monte Carlo tree searches in the field of automatic driving to solve motion planning. We propose and implement the framework of SP-MCTS based on reachable sets (continuous space), which are used to solve macro-driving strategy planning for autonomous driving. We assume that the orientation of the vehicle is always consistent with the reference path, which simplifies the definition of the action, and slices the acceleration interval as the primitive actions, and there is an intersection between the slices. In the simulation stage of MCTS, the vehicles will be classified according to the relative safety distance and make sequential decisions according to the influence relationship to achieve the interaction between the ego vehicle and the surrounding vehicles. The ego vehicle uses a heuristic random selection to determine the action in the simulation. The heuristic fully considers the information of the road and surrounding vehicles. The vehicles in the environment are simulated using IDM according to the classification or based on the assumption that they move at a constant speed. Actions are evaluated by sampling. The main application scenarios of this algorithm are one lane scenario and intersection scenario. The calculated interaction-aware reachable sets can show the maneuvering strategy in response to the scenario, which is a benefit for trajectory planning.

In the chapter discussion 5, we mentioned two points, the vehicle behavior model and action discretization, which can optimize the performance of the algorithm. Our algorithmic framework is mainly for longitude direction movement because we use the curvilinear coordinate system and reachable sets. There is insufficient research on algorithms for lateral maneuvering strategies such as lane change. In future work, one can try to add actions in the lateral direction to make the interaction-aware reachable sets represent more driving strategies. In this way, the algorithm can be applied in more scenarios. The setup of this project indicates that the behavior of the surrounding vehicles cannot be controlled, so we only make interaction-aware sequence decisions during the simulation phase of MCTS. In the future, as the proportion of self-driving vehicles in road traffic increases, the cooperation between vehicles can effectively improve traffic safety and efficiency. Then it is very valuable to study and calculate the cooperate reachable set.

List of Figures

List of Tables

Bibliography

[AAA23] Okan Asik, Fatma Başak Aydemir, and Hüseyin Levent Akın. Decoupled monte carlo tree search for cooperative multi-agent planning. *Applied Sciences*, 13(3):1936, 2023.

[AKM17] Matthias Althoff, Markus Koschi, and Stefanie Manzinger. Commonroad: Composable benchmarks for motion planning on roads. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 719–726. IEEE, 2017.

[AUK18] Matthias Althoff, Stefan Urban, and Markus Koschi. Automatic conversion of road networks from opendrive to lanelets. In *2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, pages 157–162. IEEE, 2018.

[AW20] Matthias Althoff and Gerald W‘ursching. Commonroad: Vehicle models. 2020.

[BEHK20] Julian Bernhard, Klemens Esterle, Patrick Hart, and Tobias Kessler. Bark: Open behavior benchmarking in multi-agent environments. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6201–6208. IEEE, 2020.

[BPW+12] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[Cha10] Guillaume Maurice Jean-Bernard Chaslot Chaslot. *Monte-carlo tree search*, volume 24. Maastricht University, 2010.

[Cou07a] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers 5*, pages 72–83. Springer, 2007.

[Cou07b] Rémi Coulom. Monte-carlo tree search in crazy stone,“. In *Proc. Game Prog. Workshop, Tokyo, Japan*, pages 74–75, 2007.

[CPW12] Peter I Cowling, Edward J Powley, and Daniel Whitehouse. Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):120–143, 2012.

[CWH+08] Guillaume M Jb Chaslot, Mark HM Winands, H Jaap van den Herik, Jos WHM Uiterwijk, and Bruno Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357, 2008.

[FHL08] Dave Ferguson, Thomas M Howard, and Maxim Likhachev. Motion planning in urban environments. *Journal of Field Robotics*, 25(11-12):939–960, 2008.

[Gib97] Robert Gibbons. An introduction to applicable game theory. *Journal of Economic Perspectives*, 11(1):127–149, 1997.

[GPMN15] David González, Joshué Pérez, Vicente Milanés, and Fawzi Nashashibi. A review of motion planning techniques for automated vehicles. *IEEE Transactions on intelligent transportation systems*, 17(4):1135–1145, 2015.

[GVM+10] Sébastien Glaser, Benoit Vanholme, Saïd Mammar, Dominique Gruyer, and Lydie Nouveliere. Maneuver-based trajectory planning for highly autonomous vehicles on real road with traffic and driver interaction. *IEEE Transactions on intelligent transportation systems*, 11(3):589–606, 2010.

[HMXB17] Elwan Héry, Stefano Masi, Philippe Xu, and Philippe Bonnifait. Map-based curvilinear coordinates for autonomous vehicles. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7. IEEE, 2017.

[KEZ18] Karl Kurzer, Florian Engelhorn, and J Marius Zöllner. Decentralized cooperative planning for automated vehicles with continuous monte carlo tree search. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 452–459. IEEE, 2018.

[KF11] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[KJ18] Hunter Schafer Nathaniel Yazdani Kevin Jamieson, Christopher Mackie. Lecture 19: Monte carlo tree search. *CSE599i: Online and Adaptive Machine Learning*, Apr 2018.

[Kor85] Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[KTH10] Arne Kesting, Martin Treiber, and Dirk Helbing. Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity. *Philosophical*

Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 368(1928):4585–4605, 2010.

[KZZ18] Karl Kurzer, Chenyang Zhou, and J Marius Zöllner. Decentralized cooperative planning for automated vehicles with hierarchical monte carlo tree search. In *2018 IEEE intelligent vehicles symposium (IV)*, pages 529–536. IEEE, 2018.

[LKK16] David Lenz, Tobias Kessler, and Alois Knoll. Tactical cooperative planning for autonomous highway driving using monte-carlo tree search. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 447–453. IEEE, 2016.

[LW20] Andres Ladino and Meng Wang. Game theoretic lane change strategy for cooperative vehicles under perfect information, 2020.

[LWKA22] Edmond Irani Liu, Gerald Würsching, Moritz Klischat, and Matthias Althoff. Commonroad-reach: A toolbox for reachability analysis of automated vehicles. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2313–2320. IEEE, 2022.

[MPA20] Stefanie Manzinger, Christian Pek, and Matthias Althoff. Using reachable sets for trajectory planning of automated vehicles. *IEEE Transactions on Intelligent Vehicles*, 6(2):232–248, 2020.

[MUDL11] Matthew McNaughton, Chris Urmson, John M Dolan, and Jin-Woo Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *2011 IEEE International Conference on Robotics and Automation*, pages 4889–4895. IEEE, 2011.

[PČY+16] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

[SA17] Sebastian Söntges and Matthias Althoff. Computing the drivable area of autonomous road vehicles in dynamic road scenes. *IEEE Transactions on Intelligent Transportation Systems*, 19(6):1855–1866, 2017.

[ŚGSM22] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, pages 1–66, 2022.

[SM11] Leandro Soriano Marcolino. *Multi-agent monte carlo go*. PhD thesis, Future University Hakodate, 2011.

[SWT09] Alexander Shkolnik, Matthew Walter, and Russ Tedrake. Reachability-guided sampling for planning under differential constraints. In *2009 IEEE International Conference on Robotics and Automation*, pages 2859–2865. IEEE, 2009.

[SWUVDH07] Jahn-Takeshi Saito, Mark HM Winands, Jos WHM Uiterwijk, and H Jaap Van Den Herik. Grouping nodes for monte-carlo tree search. In *Computer Games Workshop*, pages 276–283, 2007.

[SWVDH+08] Maarten PD Schadd, Mark HM Winands, H Jaap Van Den Herik, Guillaume MJ B Chaslot, and Jos WHM Uiterwijk. Single-player monte-carlo tree search. In *Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29-October 1, 2008. Proceedings 6*, pages 1–12. Springer, 2008.

[THNS89] Arata Takahashi, Takero Hongo, Yoshiki Ninomiya, and Gunji 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*, pages 392–397. IEEE, 1989.

[TLW14] Mandy JW Tak, Marc Lanctot, and Mark HM Winands. Monte carlo tree search variants for simultaneous move games. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.

[WA21] Gerald Würsching and Matthias Althoff. Sampling-based optimal trajectory generation for autonomous vehicles using reachable sets. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 828–835. IEEE, 2021.

[ZA23] Christoph Ziegler and Jürgen Adamy. Anytime tree-based trajectory planning for urban driving. *IEEE Open Journal of Intelligent Transportation Systems*, 2023.

License