Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Master Thesis

Masterarbeit

Parallel Priority-Based Trajectory Planning with Safety Guarantees for Networked Vehicles

Parallele Prioritätsbasierte Trajektorienplanung mit Sicherheitsgarantien für Vernetzte Fahrzeuge

Jianye Xu
Matrikelnummer: 409851

Aachen, September 19, 2022

Examiners:
Prof. Dr.-Ing. Stefan Kowalewski

Advisors:
Patrick Scheffe M.Sc.

This thesis was submitted to
Lehrstuhl Informatik 11 – Embedded Software

Acknowledgments

First of all, I would like to thank my examiner, Prof. Dr.-Ing Stefan Kowalewski, who made it possible to write a thesis as a student of control engineering at his Chair of Computer Science 11 - Embedded Software, RWTH Aachen University. In addition, I would like to thanks for his remarks on my work.

This work would not have been possible without my advisor Patrick Scheffe. I would like to express my deepest gratitude to him. His dedicated guidance, critical insights, and inspiring ideas helped me to complete this thesis to the maximum extent possible and made this experience memorable!

Besides, I am very grateful to my teacher Dr.-Ing. Bassam Alrifaee, who aroused my interest in networked control systems and gave me continuous encouragement. I would also like to thank David Becher, Bingyi Su, Julius Kahle, Georg Dorndorf, and Ole Greß for their help and discussions when I encountered difficulties, and Keith Cherayi for reviewing parts of this thesis.

Finally, I would like to give my sincere thanks to my family for their enduring support, which brought me into the position of writing this thesis, and my girl friend for her understanding when I had hard times.

Abstract

This master's thesis focuses on priority-based trajectory planning in large-scale Networked Control Systems (NCSs), where agents are prioritized, and lower-priority agents are expected to avoid collisions with higher-priority agents. According to trajectory planning order, sequential and parallel trajectory planning are classified. In sequential trajectory planning, the total trajectory planning time accumulates, which impairs its application in large-scale NCSs. Parallel trajectory planning, by contrast, does not have this problem since agents plan simultaneously at each time step. However, it intrinsically suffers from the prediction inconsistency problem – the assumed trajectory of one agent by another agent is inconsistent with the actual trajectory, which may result in collisions even if this agent could find the optimal solution for its trajectory optimization problem. This work proposes a strategy to address this problem using reachability analysis, i.e., letting lower-priority agents avoid the reachable sets of higher-priority agents. The high computation time of calculating the reachable sets of nonlinear dynamic systems is reduced considerably using offline calculated reachable sets. This strategy provably guarantees safety. However, lower-priority agents behave conservatively since they should avoid all the reachable sets of agents with higher priorities. A Model Predictive Control (MPC)-based framework combining sequential and parallel trajectory planning is presented to mitigate this problem, letting highly coupled agents form groups and plan trajectories sequentially. This agent grouping problem is converted to a graph partitioning problem by estimating the coupling degrees between agents. The simulative experiments show that the proposed strategy effectively solves the prediction inconsistency problem, leading to an infinite collision-free runtime. In addition, they also show that the proposed framework provides an effectual way to reduce the conservativeness of the proposed strategy.

Contents

1. Introduction

The repaid development of communication technologies brings life to Networked Control Systems (NCSs). NCSs are spatially distributed systems where communications between sensors, actuators, and controllers occur [ZHG+20]. They have been applied to many fields, such as aircraft, industrial automation, and automobiles. The control of NCSs has been widely researched in the recent years, and this thesis focuses on the predictive control in NCSs, or specifically, trajectory planning in NCSs.

Model Predictive Control (MPC), also known as Receding Horizon Control (RHC), is an advanced control method with three main ideas behind it [CA13, Sec. 1.1]:

1. Explicit use of models.
2. Calculate control inputs while minimizing some objective values and considering some constraints.
3. Use receding horizon to consider future performance.

Those ideas make MPC an ideal candidate for solving networked control problems in NCSs, where, for example, collisions between agents must be avoided while operating. According to [Alr17, Sec. 3.4], networked MPC can be classified into three types: Centralized MPC (CMPC), decentralized MPC, and Distributed MPC (DMPC).

CMPC is characterized by its unique center controller, which has global knowledge about all agents and is thus able to find the optimal trajectories if they exist [Alr17]. However, its time complexity is exponential in the number of agents [KKM+19, BBT02], which in turn makes CMPC impractical for NCSs. Another drawback of CMPC is the single point of failure since only one (center) controller exists [Alr17]. In decentralized MPC, each agent has its own controller, planning its trajectory with considering its own object [Alr17, Sec. 3.4]. It alleviates the computation effort of CMPC, but one downside lies within the trajectory quality since agents do not communicate with each other, limiting its application range in NCSs where agents coordinate through communication networks [GYH17]. In DMPC, each agent also has its own controller and can communicate with others, thus making it possible to solve trajectory optimization problems while considering a subset of agents. Similar to decentralized MPC, DMPC has good scalability with the number of agents [KKM+19].

DMPC could be realized either in a cooperative or a non-cooperative way. In the cooperative way, each agent plans trajectories not only for itself but also for

a set of other agents. Thereafter, they exchange their planned trajectories and reach consensuses [SKA22]. In the non-cooperative way, each agent plans its own trajectory, and if necessary, they communicate their trajectories so that some agents can consider other agents' planned trajectories to avoid collisions. To determine which agents are responsible for collision avoidance, one can assign priorities to them, i.e., lower-priority agents are expected to avoid collisions with higher-priority agents. This approach is called priority-based non-cooperative DMPC (P-DMPC) and will be further explained.

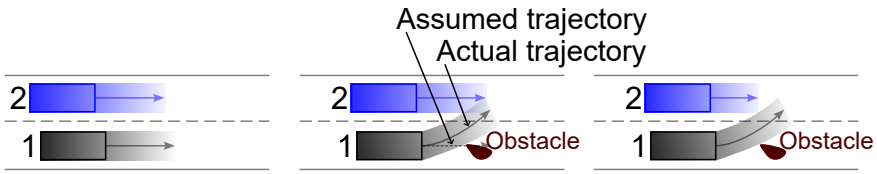
In P-DMPC, agents plan trajectories either in sequence or parallel. In sequential trajectory planning, agents plan trajectories one after another. Higher-priority agents plan trajectories earlier and then communicate their plans to lower-priority agents so as to avoid collisions. The total trajectory planning time of each time step is the sum of the trajectories planning times of all vehicles. Compared with sequential trajectory planning, agents plan their trajectories simultaneously in parallel trajectory planning at each time step. Roughly speaking, the total trajectory planning time is irrelevant to the number of agents.

1.1. Motivation

If sequential trajectory planning is used in P-DMPC, the total trajectory planning time will increase with the number of agents, which impairs its application in large-scale NCSs. That can be addressed using parallel trajectory planning. However, the prediction inconsistency problem arises, i.e., the assumed trajectory of one agent by another agent is inconsistent with the actual trajectory of this agent. In this case, the lower-priority agents can only avoid the assumed trajectories of higher-priority vehicles, which may deviate from their actual trajectories. Fig. 1.1 illustrates the risk of this problem. Assume vehicle 1 has a higher priority than vehicle 2. In (a), two vehicles plan their trajectories in parallel at time step k. They all decide to move at a moderate pace. At the next time step (Fig. 1.1(b)), vehicle 1 changes its plan and turns left to avoid a suddenly detected obstacle. However, vehicle 2 assumes that vehicle 1 will keep using its previous plan, which is go straight. Vehicle 2 thus plans to keep its speed instead of braking. As one can see, there is a collision risk as their planned trajectories intersect, i.e., vehicle 2 with a lower priority endangers vehicle 1. This problem disappears if they plan trajectories one after another since vehicle 2 can adapt its speed according to the actual trajectory of vehicle 1 (see Fig. 1.1(c)).

1.2. Main contributions

The contribution of this thesis is twofold.



(a) Time step k. (b) Time step k+1: Plan in parallel. (c) Time step k+1: Plan in sequence.

Figure 1.1.: The prediction inconsistency problem.

1. The prediction inconsistency problem between parallel planning agents is addressed using reachability analysis, which, to the best of the author's knowledge, has never been used in this way before. A computationally fast method is presented to calculate the reachable sets of agents with nonlinear dynamics, which largely divides the online computation burden into offline calculations by exploiting the geometric invariance property of the transformation of states of many mechanical systems, such as vehicles and robots [FDF05, FOW19].
2. An MPC-based framework combining sequential and parallel trajectory planning for NCSs is proposed, which provides a way to balance the conservativeness of the proposed strategy dealing with the prediction inconsistency problem and the trajectory planning time. With provable safety guarantees, a strategy dealing with infeasibilities – that is, agents cannot find feasible trajectories avoiding collisions – is proposed by exploiting the nature of MPC.

1.3. Outline

This thesis is structured into six chapters. The first chapter introduces this thesis's background, motivation, and contributions. Chapter 2 gives the basics that are required to understand this thesis. Chapter 3 summarizes the related work on assigning priority, dealing with the prediction inconsistency problem, combining sequential and parallel trajectory planning, partitioning graphs, and calculating reachable sets. The core chapter, Chapter 4, presents the proposed MPC-based framework detailedly. Chapter 5 evaluates the framework. Chapter 6 draws the final conclusions and outlooks.

2. Basics and Requirements

The chapter introduces basic concepts and preliminary knowledge required to understand this thesis. Section 2.1 introduces firstly some basic definitions of graph theory and then the min-cut clustering problem. Section 2.2 gives basics of networked control systems, and how are they modeled by graphs. Section 2.3 explains what is reachability analysis. Section 2.4 details how to convert a trajectory planning problem into a graph search problem. In the last section, Section 2.5, assumptions regarding to agents, hardware, and communication are given.

2.1. Graph Theory

Graph theory will be used to model NCSs. Firstly, some basics of graph theory are introduced (mainly based on [Alr17, Sec. 2.2] and [Bol12]). In Section 2.1.2, a classical graph partitioning problem, the min-cut clustering problem, is described.

2.1.1. Basic Definitions

Definition 1 (Graph)**.** *Graphs are mathematical structures to model pairwise relations between objects. A graph consists of vertices (also called nodes) and edges connecting part of vertices. For example, the edge $e_{i,j}$ connects vertices v_i and v_i. Formally, this can be expressed by $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \ldots, v_N\}$ is a set of vertices and $\mathcal{E}$ is a set of edges.*

The number of vertices is $N = |\mathcal{V}| \in \mathbb{N}$ and the number of edges is $M = |\mathcal{E}| \in \mathbb{N}$. Two vertices v_i, v_j are adjacent or are neighbor if they are connected by an edge $e_{i,j} \in \mathcal{E}$. All neighbors of vertex v_i is denoted by $\mathcal{V}^{(i)} = \{v_{i_1}, \ldots, v_{i_{N^{(i)}}}\}$ with $N^{(i)}$ being the number of neighbors:

$$\mathcal{V}^{(i)} = \{v_j \mid \forall v_j \in \mathcal{V} : j \neq i, (v_i, v_j) \in \mathcal{E}\}.$$

Operator $G(\mathcal{V}) : \mathcal{V} \to G$ returns the corresponding graph of vertices $\mathcal{V}$. Analog, $\mathcal{V}(G) : G \to \mathcal{V}$ returns the vertices of the graph G.

Next, path and cycle are defined.

Definition 2 (Path). *A path of a graph G is a subgraph $G_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi) \subseteq G$ with distinct vertices $\mathcal{V}_\pi = \{v_1, v_2, \ldots, v_{l-1}, v_l\}$ and distinct edges $\mathcal{E}_\pi = \{(v_1, v_2), \ldots, (v_{l-1}, v_l)\}$. The length of G_π is defined as $|\mathcal{E}_\pi| = l - 1$.*

Definition 3 (Cycle). *A path is called a cycle if it starts from a vertex and ends at the same vertex.*

Two types of graphs are distinguished according to whether their edges are directed or not: undirected graphs and directed graphs.

Definition 4 (Undirected graph). *In undirected graphs, edges are undirected, i.e., $e_{i,j} = e_{j,i}$. Formally, $\mathcal{E} \subseteq \{\{v_i, v_j\} \mid v_i, v_j \in \mathcal{V}, v_i \neq v_j\}$ is a set of unordered pairs of vertices. A undirected graph is connected if at least one (undirected) path exists between any two distinct vertices.*

Definition 5 (Directed graph). *In directed graphs, edges are directed, i.e., $e_{i,j} \neq e_{j,i}$. Each edge points from its start vertex to the end vertex. Formally, $\mathcal{E} \subseteq \{(v_i, v_j) \mid (v_i, v_j) \in \mathcal{V}^2, v_i \neq v_j\}$ is a set of ordered pairs of vertices. A directed graph is strongly connected if at least one directed path exists between each pair of distinct vertices. It is weakly connected if replacing all its edges with undirected edges yields a connected undirected graph.*

In directed graphs, the in-degree $\delta_{in}^{(i)}$ of a vertex i is the number of edges incoming to it while the out-degree $\delta_{out}^{(i)}$ is the number of edges outgoing from it. The incoming (outgoing) edges are called in-edges (out-edges) of it.

Tree, Forest, Directed Acyclic Graph

A graph is acyclic if it contains no cycle. An acyclic undirected graph is called a tree if it is connected; otherwise, it is called a forest (a disjoint union of multiple trees). A directed graph is called a Directed Acyclic Graph (DAG) if it contains no cycle. Analog, a DAG is called a directed tree if it is weakly or strongly connected; otherwise, it is called a directed forest. A directed forest consists of multiple components. A component is called a weakly connected component if it is weakly connected.

Root, Leaf, Level, Depth

In a DAG, vertices whose in-degree is zero are called roots, while whose out-degree is zero are called leaves. The level of a vertex is the maximum length among directed paths from roots to this vertex, e.g., the level of a root is zero. The operator $d(G) : G \to \mathbb{N}$ returns the depth of a DAG G, which is the length of the longest path in G or alternatively, the maximum level of its leaves. Fig. 2.1 depicts a directed

tree with eight vertices, where two roots and two leaves are shown in red and blue circles, respectively. The depth of the underlying DAG is five as the longest path is $G_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ with $\mathcal{V}_\pi = \{1, 2, 4, 5, 7, 8\}$.

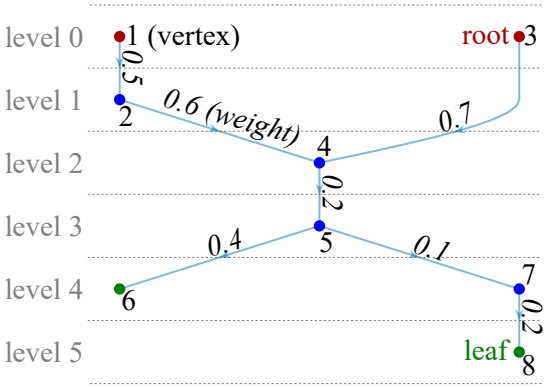


Figure 2.1.: A directed Tree with eight vertices.

Weight

Each edge is associated with a numerical value in many applications, called a weight w_e. The resulting graph is called an edge-weighted graph. Edge-weighted graphs can be either undirected or directed. The weighting matrix $A_w \in \mathbb{R}^{N \times N}$ contains weights between each pair of vertices, e.g., $A_w(i, j)$ is the weight between vertices i and j, which is denoted as $w_e^{(i,j)}$. The operator $A_w(G) : G \to \mathbb{R}^{N \times N}$ returns the weighting matrix of the graph G.

2.1.2. Min-cut Clustering Problem

The problem of clustering arises in many fields, such as data mining, social sciences, and traffic systems. In this thesis, agents are to be grouped. The min-cut clustering problem is a subclass of graph partitioning problems, the objective of which is to divide the given graph into multiple subgraphs. This can be formally formulated by Problem 1.

Problem 1 (Min-cut clustering problem)**.** *Given an edge-weighted DAG $G = (\mathcal{V}, \mathcal{E})$ with edge weight w_e for each edge $e \in \mathcal{E}$, find a partition $\Gamma = \{\mathcal{V}_1, \ldots, \mathcal{V}_k\}$ of $\mathcal{V}$ that solves 2.1a while satisfying constraints from 2.1b to 2.1c.*

$$\Gamma = \arg\min_{\Gamma} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \sum_{e \in \mathcal{E}_{i,j}} w_e \tag{2.1a}$$

subject to

$$\mathcal{V}_1 \cup \mathcal{V}_2 \cup \cdots \cup \mathcal{V}_k = \mathcal{V}, \tag{2.1b}$$

$$\mathcal{V}_i \cap \mathcal{V}_j = \emptyset, \forall i, j \in \{1, \ldots, k\}, i \neq j. \tag{2.1c}$$

Here, k is the number of resulting subgraphs. Note that empty subgraphs are allowed here. If this number should not be predefined, one can set k to the total number of vertices. As a result, the actual number of subgraphs equal the number of nonempty subgraphs. $\mathcal{E}_{i,j}$ denotes the cut edges between subgraph i and j, that is, a set of edges that connect one vertex in subgraph i and one vertex in subgraph j:

$$\mathcal{E}_{i,j} = \{\{v_i, v_j\} \in \mathcal{E} \mid v_i \in G_i, v_j \in G_j\}.$$

If $i = j$, the denotation $\mathcal{E}_{i,j}$ will be simplified to $\mathcal{E}_i$, meaning a set of edges inside subgraph i.

Each vertex corresponds to one agent, and each subgraph corresponds to one group of agents. In this way, different groups of agents can be directly formed after finding a solution of Problem 1. Constraint 2.1b and 2.1c ensure that each vertex belongs to exactly one subgraph; that is, no agent will be in multiple groups at the same time.

Note that Problem 1 is equivalent to a max-cut clustering problem [JMN93]. The objective function 2.1a minimizes the sum of weights of edges crossing subgraphs, which equals to maximize the sum of weights of edges within each subgraph since the sum of weights of all edges is constant.

2.2. Networked Control Systems

In this section, the basics of NCSs are introduced, which are mainly based on [Alr17, Sec. 3.2].

NCSs can be modeled by graphs, where each vertex corresponds to an agent. There is an edge connecting two vertices if the corresponding agents are coupled. The term *coupled* is conceptualized as follows (note that is will be specialized in Definition 16):

Definition 6 (Coupled)**.** *Two agents i and j are coupled if and only if collision possibility exists in a specific time.*

Next, a coupling graph is defined, which models coupling between agents in a NCS.

Definition 7 (Coupling graph)**.** *A coupling graph models the interaction between agents represented by distinct vertices. An edge connecting two vertices indicates a coupling objective or constraint between the two corresponding agents. In the case of a directed edge, the coupling objective or constant is only associated with the end vertex. The coupling graph is called a directed coupling graph if all edges are directed.*

The coupling situation of a NCS will be described by the so-called coupling matrix $A_c \in \mathcal{R}^{N\times N} : \mathcal{V} \times \mathcal{V} \to \{0, 1\}$. $A_c(i, j) = 1$ means that agent i is coupled with agent j, while $A_c(i, j) = 0$ indicates no coupling. The operator $A_w(G) : G \to \mathcal{R}^{N\times N}$ queries the coupling matrix of the graph G.

Definition 8 (Weakly coupled)**.** *Two agents are weakly coupled if they are in the same connected component in their undirected coupling graph or in the same weakly connected component of their directed coupling graph.*

The set of weakly connected agents with agent i will be denoted as $\mathcal{V}_{wc}^{(i)}$.

Next, the coupling weight indicating the coupling degree between two agents is defined.

Definition 9 (Coupling weight)**.** *The coupling weight w_e is a scalar value indicating the coupling degree of two coupled agents. A higher value indicates a stronger coupling. The coupling weight of two agents is zero if they are uncoupled.*

In NCSs, passive agents and active agents are distinguished.

Definition 10. *Passive agents represent dynamic systems without networked control. They can, however, communicate with active agents or be detected by their sensors.*

Definition 11. *Active agents are controlled in NCSs. They are connected by communication networks, making it possible to interact with others.*

A set of coupled agents of agent i is denoted by $\mathcal{V}^{(i)} \in \mathbb{N}$ while $\mathcal{V}_p^{(i)} \in \mathbb{N}$ represents a set of passive agents relevant to it, i.e., all safety-related static obstacles in the environment such as road infrastructures.

The computation time T_{ct} of a NCS is a deciding feature to judge if the used control strategy is real-time or not.

Definition 12 (Computation time (from Definition 3.8 of [Alr17]))**.** *The computation time T_{ct} of a NCS is the whole time required for all agents to find a solution at a given time step. That includes measuring, communicating, trajectory planning, and applying control inputs.*

The trajectory planning time is a part of the computation time and will be denoted as $T_{ct,traj}$.

The computation level of an agent is an important parameter and is defined as follows.

Definition 13 (Computation level)**.** *In priority-based trajectory planning, each agent has a computation level deciding its trajectory planning order. The lower an agent's computation level, the higher the priority, and thus the earlier it plans its trajectory.*

2.3. Reachability Analysis

This section introduces the concept of reachability analysis.

Firstly, the dynamic of a nonlinear system is defined by the ordinary differential equation

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}). \tag{2.2}$$

The configuration space $\mathcal{X} \in \mathbb{R}^{n_x}$ is the possible set of the system states $\boldsymbol{x}$, and the input space $\mathcal{U} \in \mathbb{R}^{n_u}$ is the possible set of the control inputs $\boldsymbol{u}$. Here, n_x and n_u are the numbers of states and control inputs. Bold letters are used to indicate vectors. Note that this rule applies only to the system states and control inputs.

Path Γ and trajectory π are distinguished: A path consists of a set of points while a trajectory order those points in time. An agent's path or trajectory refers to the path or trajectory of its center of gravity. During motion planning, agents are required to follow their reference trajectories, given by their high-level route planners, as close and fast as possible. To avoid collisions with others, each agent should at least avoid the occupied sets of others. Here is the definition of the occupied set of an agent:

Definition 14 (Occupied set (adapted from Definition 1 of [PA21]))**.** *The occupied set of an agent is the footprint of its moving along its trajectory.*

The operator $\mathcal{O}(\pi) : \mathbb{R}^2 \to \mathbb{R}^2$ returns the occupied set of an agent based on its trajectory π, and $\mathcal{O}(\pi^{(i)}_{([t_a,t_b])})$ describes the occupied set of agent i of a time interval $[t_a, t_b]$. Next, the reachable set of an agent is defined.

Definition 15 (Reachable set (adapted from Definition 4 of [AM16]))**.** *The reachable set of agent i at time $t = r$ is*

$$\mathcal{R}^{(i)}_{(r)} = \{\int_0^r f(\boldsymbol{x}, \boldsymbol{u})dt \mid \boldsymbol{x}_{(0)} \in \mathcal{X}_{(0)}, \forall t : \boldsymbol{u}_{(t)} \in \mathcal{U}\}, \tag{2.3}$$

and its reachable set of a time interval $t \in [0, r]$ *is denoted by*

$$\mathcal{R}^{(i)}_{([0,r])} = \bigcup_{t\in[0,r]} \mathcal{R}^{(i)}_{(t)}. \tag{2.4}$$

Note that if the $\boldsymbol{x}$ in Equation 2.3 refers to the states of an agent's center of gravity (usually the case), the determined set is the reachable set of its center of gravity. Most of the time, the reachable occupied set of an agent is of interest, that is, the occupied set that is reachable by its body. It can be determined based on the reachable set of its center of gravity and its geometrical dimensions. In the remaining of this thesis, the term reachable set(s) always refers to reachable occupied set(s) if not explicitly stated. Furthermore, the reachable set of agent i for n_k time steps at time step k is denoted by $\mathcal{R}^{(i)}_{([k,k+n_k]|k)}$.

2.4. Graph-Based Trajectory Planning

Trajectory planning is one of the most critical parts in NCSs. This section introduces the idea of graph-based trajectory planning (mainly based on [SPFA21]). Firstly, Section 2.4.1 presents the receding horizon control, which serves as the primary control framework. Next, Section 2.4.2 introduces the Motion Primitive Automaton (MPA) that converts the trajectory planning problem of nonlinear systems into a graph search problem. Section 2.4.3 shortly introduces the graph search algorithm used in this master's thesis.

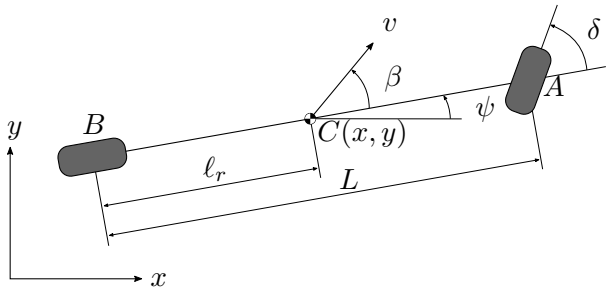


Figure 2.2.: Kinematic bicycle model.

2.4.1. Receding Horizon Control

RHC is employed as the primary control framework. Note that RHC and MPC are used interchangeably in this thesis. There are many different RHC algorithms, but all of them have three common elements [CA13, Sec. 2.1]:

1. Predicted model.
2. Objective function.
3. Control law.

The following three subsections will describe those three elements of the RHC used in this thesis.

Predicted Model

The standard nonlinear kinematic single-track model (also known as the kinematic bicycle model, see [Raj11, Sec. 2.2]) is the predicted model, where both longitudinal and lateral dynamics can be considered. In kinematic single-track model, only one front and one rear wheel are considered, as roll dynamics are ignored. Pitch dynamics are also ignored since vehicles are assumed to have planar motion. Fig. 2.2 visualizes the used model with the following notations:

- A, B, C: front wheel, center of gravity, rear wheel
- $x, y \in \mathbb{R}$: x- and y-coordinate
- $\ell_r, L \in \mathbb{R}$: rear wheel base, wheel base
- $\beta, \psi, \delta \in [0, 2\pi)$: slip angle, yaw angle, steering angle
- $v \in \mathbb{R}$: speed

The equation of motion is given as

$$\begin{cases} \dot{x} = v \cdot \cos(\psi + \beta) \\ \dot{y} = v \cdot \sin(\psi + \beta) \\ \dot{\psi} = v \cdot \frac{1}{L} \cdot \tan(\delta) \cdot \cos(\beta) \\ \dot{v} = u_v \\ \dot{\delta} = u_\delta \end{cases} \tag{2.5}$$

with

$$\beta = \tan^{-1}(\frac{\ell_r}{L} \tan(\delta)). \tag{2.6}$$

The system states can be written as a vector containing five entries

$$\boldsymbol{x} = (x, y, \psi, v, \delta)^T \in \mathbb{R}^5, \tag{2.7}$$

and the system inputs are

$$\boldsymbol{u} = (u_v, u_\delta)^T \in \mathbb{R}^2. \tag{2.8}$$

Objective Function

The objective function is used to determine the optimal control inputs. The general idea is to find the control inputs for a predefined horizon (known as control horizon, H_u) such that the sum of the objective values in a predefined horizon (called prediction horizon, H_p) is minimum. Let us specialize the term *coupled* in Definition 6.

Definition 16 (Coupled)**.** *Two agents i and j are coupled at time step k if and only if collision possibility exists in the prediction horizon H_p.*

Next, the basic optimization problem of the networked MPC is defined (adapted from [Alr17], Sec. 3.2 and [SPFA21]).

Problem 2 (Trajectory Planning Problem)**.** *Given a 2-D road map and N agents with their reference trajectories, each agent should find a trajectory that allows them to 1) follow its reference trajectory as close as possible, 2) save as much energy as possible, and 3) avoid collisions with other agents as well as static obstacles like lane boundaries.*

Formally, for each agent $i = 1, \ldots, N$ at time step k, this can be formulated as

$$J_{(k)}^{(i)} = \sum_{h=1}^{H_p} l_x^{(i)}(\boldsymbol{x}_{(k+h)}^{(i)}, \boldsymbol{r}_{(k+h)}^{(i)}) + \sum_{h=0}^{H_u-1} l_u^{(i)}(\boldsymbol{u}_{(k+h)}^{(i)}) \tag{2.9a}$$

subject to

$$\boldsymbol{x}_{(k+h+1)}^{(i)} = f^{(i)}(\boldsymbol{x}_{(k+h)}^{(i)}, \boldsymbol{u}_{(k+h)}^{(i)}), h = 0, \ldots, H_p - 1 \tag{2.9b}$$

$$\boldsymbol{x}_{(k+h)}^{(i)} \in \mathcal{X}^{(i)}, h = 1, \ldots, H_p - 1 \tag{2.9c}$$

$$\boldsymbol{x}_{(k+H_p)}^{(i)} \in \mathcal{X}_{(H_p)}^{(i)} \tag{2.9d}$$

$$\boldsymbol{u}_{(k+h)}^{(i)} \in \mathcal{U}^{(i)}, h = 0, \ldots, H_u - 1 \tag{2.9e}$$

$$\mathcal{O}(\pi_{([k+h,k+h+1]|k)}^{(i)}) \cap \mathcal{O}(\pi_{([k+h,k+h+1]|k)}^{(j)}) = \emptyset, \forall j \in \mathcal{V}^{(i)}, h = 0, \ldots, H_p - 1 \tag{2.9f}$$

$$\mathcal{O}(\pi_{([k+h,k+h+1]|k)}^{(i)}) \cap \mathcal{O}(\pi_{([k+h,k+h+1]|k)}^{(p)}) = \emptyset, \forall p \in \mathcal{V}_p^{(i)}, h = 0, \ldots, H_p - 1, \tag{2.9g}$$

with the following definitions of variables and symbols:

- $l_x^{(i)}$: $\mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \to \mathbb{R}$, a penalty function for the trajectory tracking error of agent i, where n_x is the number of the related states. Here, $n_x = 2$ for x- and y-coordinate.
- $l_u^{(i)}$: $\mathbb{R}^{n_u} \to \mathbb{R}$, a penalty function for the control inputs of agent i. Here, $n_u = 2$ for the acceleration in speed and the angular acceleration in steering angle.
- $\boldsymbol{r}_{(k+h)}^{(i)} \in \mathbb{R}^2$: The segment of agent i's reference trajectory during the time step $k + h$.

- $\mathcal{V}^{(i)} \in \mathbb{N}$: A set of coupled agents with agent i.
- $\mathcal{V}_p^{(i)} \in \mathbb{N}$: A set of passive agents of agent i.

Note that each subscript inside parentheses represents a moment. For simplification of notation, the subscript of the symbol $\boldsymbol{r}$ (reference trajectory) stands for the duration of this time step. For example, $\boldsymbol{r}_{(k)}$ describes the segment of the reference trajectory of the time interval $[k, k+1]$, which equals $\boldsymbol{r}_{([k,k+1])}$.

Control Law

The control inputs for each agent $i \in \{1, \ldots, N\}$ are obtained by minimizing the objective function 2.9a

$$\boldsymbol{u}^{(i)}(\cdot) = \arg\min_{\boldsymbol{u}(\cdot)} J^{(i)} \tag{2.10}$$

while fulfilling constraints from 2.9b to 2.9g, where $\boldsymbol{u}_{(\cdot)}^{(i)} = (u_{(k)}, \ldots, u_{(k+H_u-1)})^T$. At each time step, the control inputs are obtained not only for the current time step k but also for the future $H_u - 1$ time steps, whereas the control inputs of only the current time step k are applied, that is, $u_{(k)}$. Normally, the control inputs for all future time steps are discarded. However, they can be used as a fallback, i.e., agents can use the previously calculated control inputs when they cannot find feasible trajectories [GA19, Su22]. In Section 4.4.2, this idea is detailed.

In this thesis, an equal number of prediction horizon and control horizon will be used ($H_p = H_u$). The term prediction horizon is preferred in the remaining of the thesis.

2.4.2. Motion Primitive Automaton

The proposed framework is applied to car-like robots, typical mechanical systems that have symmetric property, i.e., rotating and translating invariances. For the theoretical background, readers are referred to [FDF05].

A MPA consists of trim primitives (in short, trims) and motion primitives. Trim primitives, also known as relative equilibria, are steady-state motions with constant control inputs [SPFA21]. Due to this property, trim primitives quantize system states. $\boldsymbol{x}^{t_\square}$ denotes the states representing by trim primitive $t_\square$. A transition from one trim primitive to another called a motion primitive, a class of equivalent trajectories (see Definition 4 in [SPFA21]).

Definition 17 (Transition (Adapted from Definition 3 of [SPFA21]))**.** *A transition is a finite-time trajectory of a system if its states evolve form trim t_i to trim primitive t_j. It is characterized by a duration T, a control input $\boldsymbol{u} : [0, T] \to \mathcal{R}^{n_u}$, an initial states $\boldsymbol{x}^{t_i}$ and an end states $\boldsymbol{x}^{t_j}$.*

Note that if the duration of each trim primitives equal the sample time used in MPC, exactly one motion primitive is required to build a trajectory for each step. Two trims are connected if the system states presented by one trim are allowed to switch to the system states presented by another trim. The nonlinear constraints on system dynamics can be easily considered by proper choosing of trim primitives. For example, if the maximum allowed speed of a vehicle is known, all trim primitives must have speeds lower than this value. In addition, the maximum allowed turning speed can be easily considered by letting trims with larger steering angles have lower speeds.

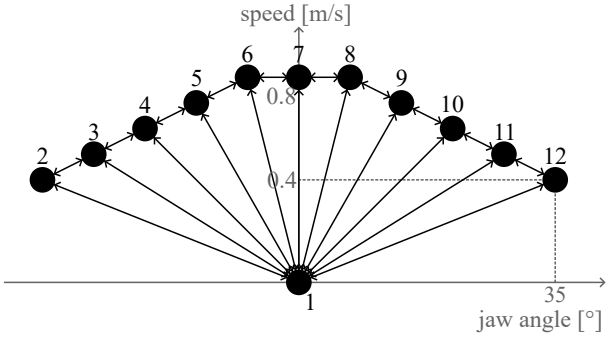


Figure 2.3.: A system with 12 trim primitives.

A MPA can be described by a directed graph where vertices present trim primitives (system states), and edges present motion primitives (system maneuvers) [FDF05]. An example is shown in Fig. 2.3. The x-coordinate presents its yaw angle while the y-coordinate presents its speed. From a trim switch to another trim is only allowed if the corresponding vertices are connected. Note that in the figure, each vertex has an edge to itself but is neglected. Based on the trim primitives, motion primitives are constructed to determine specific system maneuvers. For each trim, the transitions to its connected trims are the transferable motion primitives of this trim. The transition from a trim primitive to itself is essentially also a motion primitive. The left side of Fig. 2.4 illustrates how a transferable motion primitive of trim 8 in Fig. 2.3 is constructed. The local (dashed) polygons in Fig. 2.4 (indicated by the *local* inside parentheses) contain no pose information the vehicle, while this information is contained in the global (solid) polygons. The transition from trim 8 to one of its connected trim (trim 9) is depicted in purple dotted line, and the occupied set of this transition is approximated by a nonconvex polygon shown in red dashed line. A small inflation is added to cover the real occupied set of the transition. After that, the vehicles's actual (global) trajectory from one trim to another trim is obtained by simply translating and rotating the corresponding motion primitive. As shown on the right side of Fig. 2.4, the vehicle is currently at trim 8, and its

position is $x = 0.5, y = -0.2$ while its yaw angle is $\psi = 45°$. Suppose that this vehicle's next trim is 9. The trajectory of this transition can be calculated by firstly translating the purple dotted line to the point $[0.5, -0.2]$ and then rotating it by $45°$ counterclockwise, called a translated motion primitive. The result is depicted in purple solid line. The occupied set of this transition is obtained similarly and depicted in red solid polygon.

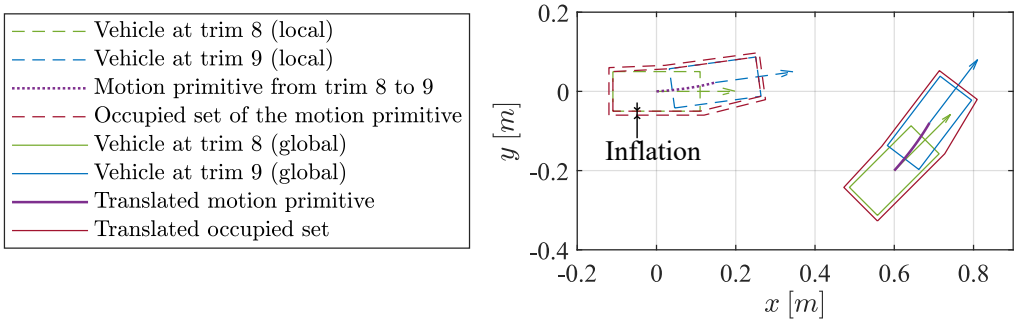


Figure 2.4.: Construct motion primitives for the system with 12 trim primitives shown in Fig. 2.3. Dashed polygons are local occupied sets of a vehicle. Solid polygons are actual (global) occupied sets.

2.4.3. Graph Search Algorithm

In Section 2.4.2, the system states and inputs are quantized. By the quantization, which makes it possible to convert the trajectory planning problem into a graph search problem. Trajectories are built by concatenating the offline built motion primitives during online planning using some efficient search-based algorithms. Interested readers are referred to [RSW04, DS16, MSS18, SPFA21]. The quantization of the system essentially facilitates the analysis of nonlinear systems.

The classical A* search algorithm is used in this thesis, which spans a spatiotemporal and hierarchical tree along the driving area. Each tree node corresponds to a trim and has two costs: cost-to-come and cost-to-go. The cost-to-come is the cost along the path to the current node, while the cost-to-go estimates the potential cost of the remaining steps. Each edge connecting two nodes corresponds to a motion primitive. During the graph searching, an edge is valid if the corresponding trajectory respects all the constraints, such as lane boundaries and the occupied sets of other vehicles; otherwise, it will be discarded. At each time step, the A* search algorithm finds the path consisting of $H_p + 1$ nodes with the minimum sum of cost of its nodes, where H_p is the defined prediction horizon. The recursive feasibility is the property that once the initial states are feasible, all the future states under the control inputs are also

feasible. One way to guarantee this property is to impose a terminal cost $\mathcal{X}_{H_p}^{(i)}$ (see constraint 2.9d in Problem 2) [May14]. Specifically, the recursive feasibility of RHC is achieved if the end node of the resulting path from the graph search algorithm corresponds to an equilibrium trim [SPFA21], those are, trims with zero speed. For example, in Fig. 2.3, the equilibrium trim is trim 1, with zero speed and zero steering angle.

2.5. Assumptions

Some assumptions are required to apply the proposed MPC-based framework.

Agents

The target NCS consists of homogeneous agents, i.e., all agents have the same dimensions and the same dynamic. Note that the proposed algorithms do not lose their generalities despite this assumption. With minor adaption, the algorithms will be suitable for inhomogeneous agents.

Hardware

Each agent has its computation unit, either an onboard or a separate computation unit. In the latter case, each agent should be equipped with an onboard wireless communication unit to receive the control inputs from its computation unit. In addition, each agent can measure the position and speed of other agents.

Communication

Agents can communicate with each other, either via an onboard communication unit or a separate device. Communication delays are not considered.

3. Related Work

This chapter overviews related work on all research topics relevant to this thesis. Firstly, Section 3.1 introduces previous work dealing with the prediction inconsistency problem between parallel planning agents. As this problem will be addressed by reachability analysis in this master's thesis, state-of-the-art approaches calculating reachable sets of dynamic systems are provided in Section 3.2. Next, sequential trajectory planning will be combined with parallel trajectory planning to alleviate the conservativeness of the proposed approach dealing with the prediction inconsistency problem. For this purpose, agents will be grouped, where agents in the same group plan trajectories in sequence while the first agents of different groups plan in parallel. This grouping problem will be converted into a graph partition problem. A brief survey of graph partitioning algorithms is available in Section 3.3. In priority-based trajectory planning, one has to handle the so-called incompleteness problem when assigning priorities, i.e., there are scenarios where some priority assignments lead to infeasibilities while some not [ČNKS15]. Section 3.4 summarizes the recent work on priority assignment in priority-based trajectory planning. Provided that collisions can be perfectly avoided, one should pay attention to deadlocks, where agents have feasible trajectories but block others mutually. Section 3.5 presents some work on avoiding deadlocks.

3.1. Dealing with the Prediction Inconsistency Problem Between Parallel Planning Agents

Research dealing with the prediction inconsistency problem in parallel trajectory planning has been proposed in recent years. There are mainly three approaches: 1) constrain system states or control inputs, 2) reach consensuses in planned trajectories, and 3) find parallelizable agents.

Constrain System States or Control Inputs

One way to reduce this problem's impact is to constrain the deviations of system states or control inputs from their previous values. For example, the authors in [DC12] researched on the stability of the distributed receding horizon control of vehicle platoons. Here, vehicles plan trajectories in parallel. The authors used a penalty

matrix to limit the deviation of system states from their previous values when solving the optimization problem, which is called move-suppression constraint in the trajectory optimization problem. In addition, the objective function consists of a term called predecessor relative error term to penalize the deviation of vehicles from their predecessors. Although the stability is guaranteed in this way, its application is limited to platoons, e.g., it cannot guarantee collision-free of two vehicles at crossing-adjacent lanelets if they plan trajectories in parallel. The authors in [FS12] proposed a distributed predictive control algorithm for large-scale linear systems. There, at each time step, each subsystem first communicates its state and input reference trajectories to its neighbors; then, it guarantees in the optimization problem that its actual trajectories deviates less than a certain bound from them by using some constraints. However, If this algorithm is applied to NCSs, the maneuverability of agents will be impaired, reducing their responsiveness to the changeable environment.

Reach Consensuses in Planned Trajectories

Some authors tried to prevent the prediction inconsistency problem by letting agents reach consensuses in their trajectories. In [KA21a], the authors proposed an iterative synchronized cooperative distributed MPC schema. Here, agents plan trajectories in parallel and not only for themselves but also for others. This means each agent's trajectory is planned by itself and many other agents. Since each agent considers only a subset of agents, its plans are local and may not be consistent with others' plans. Next, coupled agents communicate their plans with each other. If their plans are inconsistent, they synchronize those plans by averaging them, called a distributed average of local plans. If the synchronized plans are not collision-free, the agents plan trajectories again using the synchronized plans as references. This procedure is repeated until all agents reach a consensus on their plans, so the prediction inconsistency problem no longer exists. The authors evaluated the proposed schema numerically using up to 10 agents. It showed comparable performance with centralized trajectory planning while considerably reducing its computation time. Although it was proved that the iteration process will terminate, the number of needed iterations increases as the traffic scenario becomes more complex or the number of agents grows, leading to an increasing computation time. In addition, the number of iteration times highly depends on the coupling topology used. Some topologies may lead to infeasibilities.

The idea of reaching consensuses between agents is also used in [ČNKS15, MTHH19]. In [ČNKS15], the authors proposed an asynchronous distributed and parallel trajectory planning schema. At each time step, agents plan their own trajectories simultaneously and then exchange their plans with lower-priority agents. Every time an agent detects that its trajectory will collide with that of a higher-priority agent, it

replans and communicates the new trajectory to lower-priority vehicles. This process is repeated until all agents have collision-free trajectories mutually. This schema is asynchronous in the sense that at each time step, quickly planning agents do not need to idle and wait for others to finish their planning so that they can go together into the next iteration to check whether their plans are collision-free. Instead, the detection of collisions of their planned trajectories will be executed once a new trajectory is received from higher-priority agents. In this way, the computational resources distributed among each agent can be better utilized. Thus, the overall number of iterations to reach a consistency in their trajectories can be reduced. The authors proved that their iterative schema always terminates but did not analyze the number of iterations needed. The simulation results show that the iteration number has an increasing linear trend with the number of agents. In the worst case (every time the newly received plan is not collision-free), it is the same as the strategy proposed in [Alr17, Sec. 3.5], which almost degrades to a pure sequential trajectory planning in the worst case (as will be discussed later); in the best case (their first plans are "luckily" mutually collision-free), it is a pure parallel trajectory planning.

Find Parallelizable Agents

Some literature tries to find parallelizable agents; that is, find agents that can plan their trajectories in parallel without risking the prediction inconsistency problem. The author of a notable work [Alr17, Sec. 3.5] described a priority-based non-cooperative DMPC strategy. Here, a sequential trajectory planning schema is employed, which, however, differs from the classical one where agents plan one after another. Each agent belongs to a computation level, and agents in the same computation level plan trajectories in parallel while they plan in sequence if they are in different computation levels. The agents in the same computation level are uncoupled so that they do not suffer from the prediction inconsistency problem. The author pointed out that if the disturbances of agents in the first computation level are negligible, agents in the second computation level can plan in parallel with them without causing the prediction inconsistency problem, as the agents in the first computation level have the highest priority and thus do not need to consider other agents. However, the number of parallelizable agents highly depends on the coupling situation (coupling graph) of agents. In the worst case (such as a platoon of agents), no agent can plan trajectory simultaneously without causing the prediction inconsistency problem except for the agents in the first two computation levels, degrading it almost to the classical sequential trajectory planning.

In [KRSH07, KRSH06], agents are coupled if they are inside a circle with a specific diameter. Next, it was proved that agents need to only consider their coupled agents when planning trajectories, and uncoupled agents can plan trajectories simultaneously

without causing infeasibility. To maximize the number of parallelizable agents, the authors converted the agent grouping problem into a vertex coloring problem, where each vertex represents an agent. Vertices with the same color indicate that the corresponding agents are in the same group and thus can plan their trajectories simultaneously. The objective is to use a minimum number of colors while ensuring each pair of vertices connected by an edge has different colors.

3.2. Reachability Analysis

Reachability analysis determines the set of states a system can reach. starting from a set of initial states under a set of possible inputs [Alt10, p. 21]. It has recently been used widely in the safety verification of dynamic systems. An important application is the safety verification of automated vehicles.

In autonomous driving, reachability analysis must be effective and efficient since traffic situations are changeable. The calculation of reachable sets is nontrivial. For example, in most cases, one can not calculate the exact reachable set of linear continuous systems [Alt10, p. 21]. However, one can calculate the reachable set over-approximately. One disadvantage is that while the over-approximated reachable set collides with an obstacle, the actual reachable set may not. In essence, the over-approximation must be minimized while compromising acceptable computation costs [Alt10].

3.2.1. Set Representation

The set representation has a deciding impact on the complexity of reachability analysis [SK03]. Boxes are one of the simplest set representations. For a space dimension of d, a box requires only $2d$ parameters constraining its upper and lower bounds in each dimension. They have been effectively used in many fields, especially in interval analysis [Gue09, p. 20]. However, the introduced over-approximation causes overly conservative behaviors of autonomous vehicles, i.e., many drivable areas will be considered undrivable.

Polytopes are used widely to represent sets [CK03, Fre05]. Polytopes can be defined in two ways. The first is H-representation, where a polytope is a bounded intersection of finite half-spaces (facets). The second is V-representation defining a polytope using a set of points (vertices). As polytopes do not have constant representation size, one can use arbitrarily many facets (vertices) to represent a set. For this reason, the number of facets (vertices) may grow large during computing. The author of [Fre05] proposed a method to manage the complexity of polytopes by limiting the number of facets and vertices while introducing the least over-approximation.

One downside of using polytopes is the expensive computation of Minkowski sum, a commonly used operation in reachability analysis [AFG21]. This problem is avoided by using zonotopes [Gir05]. Besides Minkowski sum, linear transformation is also an essential operation used in reachability analysis. They can be computed efficiently as zonotopes are closed under those two operations [Gir05]; that is, the operating results are still zonotopes. However, they are not closed under intersection, another operation in reachability analysis. This issue was addressed in [AK11] by introducing zonotope bundles, which refer to the uncomputed intersection of a list of zonotopes.

Ellipsoids can be used as a compact representation of sets. Each is parametrized by a center point and a positive semi-definite, symmetric matrix. Therefore, their complexity depends only on space dimensions. They have been used for linear systems [KV07] as well as for nonlinear systems [AGS13]. Reachable sets can also be represented by oriented rectangular hulls [SK03], support functions [GG08], and taylor models [HFL$^+$19].

The calculation of the reachable sets of nonlinear continuous systems is generally hard. One way to facilitate this is successively linearizing the system dynamics [ADG07, ASB08, Alt13]. However, one has to pay special attention to wrapping effect, that is, the increasing over-approximation and accumulative linearization error [Alt13]. This effect is overcome in [Alt13] by introducing polynomial polytopes and in [KA21b] by sparse polynomial polytopes. Both classes of polytopes can present nonconvex sets.

3.2.2. Reachability Analysis of Maneuver Automaton

This thesis pays particular attention to the application of reachability analysis in motion primitive automaton. A notable work [HAS14] computes the reachable sets of motion primitives when constructing motion primitive automaton, allowing the computationally expensive tasks of reachability analysis to be done offline. The connectivity between motion primitives is also checked offline. The resulting motion primitive automaton is stored in a database. During online operations, the reachable sets of other traffic participants are calculated efficiently as simple models can be used [AHG13]. Next, connectable motion primitives are concatenated to build a trajectory. The trajectory is verified if the reachable sets of the used motion primitives do not intersect with the reachable sets of other traffic participants. Note that the motion primitives in this work are parametrized, making verifying infinitely many motions within the parameter space possible.

3.2.3. Invariant Safety Verification

The strategy proposed in this thesis to deal with the prediction inconsistency problem is inspired by [PA21]. This work focuses on the safe motion planning of autonomous

vehicles and presents a real-time approach guaranteeing that no collision will be caused by autonomous vehicles. During each time step, an intended trajectory is generated by an arbitrary trajectory planner that, for example, considers the most likely trajectories of other traffic participants. Next, the time-to-react is calculated, which is here defined as the maximum time that the intended trajectory can be used such that a collision-free trajectory still exists [PA21, Definition 6]. The trajectory before the time-to-react, namely the invariable safe part of the intended trajectory, is thus verified. After that, reachability analysis is employed to verify a fail-safe trajectory over an infinite time horizon formally; that is, it allows autonomous vehicles to stay safely permanently. Safety in this context means autonomous vehicles will not endanger others actively. Collisions caused by other traffic participants cannot always be avoided. For example, in traffic jams, it is generally impossible to avoid rear-end collisions caused by vehicles behind. However, the authors in [PA21] can confidently argue that the liability for accidents lies on others. The fail-safe trajectory serves as an emergency trajectory and will be concatenated to the invariable safe part of the intended trajectory. The whole trajectory is called a provably safe trajectory, which respects all the possible legal motions of others. Note that the fail-safe trajectory will only be executed when vehicles are in safety-critical scenarios.

3.3. Graph Partitioning

Problem 1 is $\mathcal{NP}$-hard [JMN93]. For a survey on the complexity of related problems, interested readers are referred to [GJ78, DJP$^+$92]. In [JMN93], the authors described a decomposition framework employing integer programming to solve this problem. The number of resulting subgraphs k can be either fixed or varied. In [CWC13], the authors provided a linear-time algorithm to partition an undirected edge-weighted simple graph into k subgraphs. The object is to minimize the max-min ratio, defined as the maximum sum of edge weights of subgraphs divided by the minimum one. The author in [Hes] described an efficient special factorization-based graph partition algorithm to cut an undirected edge-weighted graph into k parts. The maximum number of vertices on each subgraph was restricted. This class of problem is called the l-bounded graph partitioning problem, with l being the maximum number of vertices of each subgraph. However, the proposed algorithm solves the l-bounded graph partitioning problem approximately; that is, the constraint on l cannot be strictly respected.

3.3.1. Stoer's Algorithm

The min(max)-cut problem is yielded if $k = 2$ in Problem 1, i.e., cut a given graph into two parts. A simple min-cut algorithm is proposed in [SW97]. Here this algorithm

will be detailed as it will be adapted and used in this thesis (see Section 4.3.1). For convenience, this algorithm will be referred to as the first author's last name: Stoer's algorithm. It finds the minimum cut of an undirected edge-weighted graph, where the minimum cut refers to a cut of a graph into two parts such that the sum of the weights of the cut edges is minimum. The time complexity of this algorithm is $\mathcal{O}(|\mathcal{V}| + |\mathcal{V}|log(|\mathcal{V}|))$. Note that this algorithm can also be applied to a directed graph if its asymmetric coupling matrix is turned to a symmetric one.

Stoer's algorithm (see Algorithm 1) consists of two parts: the main body *MinimumCut* (lines 1 to 13) and the helper function *MinimumCutPhase* (lines 14 to 26). *MinimumCutPhase* is a procedure to find a minimum s-t-cut of a graph and is used iteratively to find a minimum cut of this graph. The term s-t-cut originates from the flow network [FFed] with two vertices (s and t) presenting the source and sink in the flow problem, and a s-t-cut has to separate the source and sink. In graph partitioning problems, a s-t-cut divides a graph into two parts, with the s-vertex being in one part and the t-vertex being in another. A minimum s-t-cut minimizes the number of cut edges in case of an unweighted graph and the sum of weights of cut edges for an edge-weighted graph. In the *MinimumCutPhase*, a set ($\mathcal{V}_{tmp}$) is initialized in line 14 with the a-vertex being its initial element, which serves as an start point to find the next one to be added. The set $\mathcal{V}_{tmp}$ is used to store all the vertices added later. During each iteration, the vertex v, which is outside of this set and has the maximum sum of weights with it, will be found (line 16):

$$v = \arg\max_{v} \sum_{v_i \in \mathcal{V}_{tmp}} A_w(v, v_i), v \notin \mathcal{V}_{tmp}, \tag{3.11}$$

and added to $\mathcal{V}_{tmp}$ (line 17). The second last and the last added vertex is the desired s- and t-vertex, respectively. The cutting result of this *MinimumCutPhase*, called cut-of-the-phase, is obtained by separating the t-vertex. The cutting cost of each cut-of-the-phase is the sum of weights of cut edges:

$$cost = \sum_{v_i \in \mathcal{V}_1} \sum_{v_j \in \mathcal{V}_2} A_w(v_i, v_j), \tag{3.12}$$

where $\mathcal{V}_1 = \mathcal{V}/\{t\}$ is the first cut (see line 21) part and $\mathcal{V}_2 = \{t\}$ is the second cut part (see line 22). To enter into the next *MinimumCutPhase*, the t-vertex will be merged into the s-vertex, i.e., all other edges connected to them will be merged to one edge with the weights being summed, and all edges between those two vertices will be deleted. Correspondingly, the vertices $\mathcal{V}$ and the coupling matrix A_w will be updated (line 24 and line 25).

Algorithm 1: Stoer's algorithm

Input: all vertices: $\mathcal{V}$, weighting maxtrix: $A_w \in \mathcal{R}^{N \times N}$, start vertex: a
Output: vertices of the resulting subgraphs: $\mathcal{V}_1, \mathcal{V}_2$

$MinimumCut(A_w, a)$:

1 $cost_{min} \leftarrow inf$;
2 **if** a not given **then**
3 | $a \leftarrow 1$; `// defaults to the first vertex`
4 **end**
5 **while** $|\mathcal{V}| > 1$ **do**
6 | $\mathcal{V}, \mathcal{V}_{1,tmp}, \mathcal{V}_{2,tmp}, cost_{tmp} \leftarrow MinimumCutPhase(\mathcal{V}, A_w, a)$;
7 | **if** $cost_{tmp} < cost_{min}$ **then**
8 | | $cost_{min} = cost_{tmp}$; `// record the minimum cost`
9 | | $\mathcal{V}_1 \leftarrow \mathcal{V}_{1,tmp}$;
10 | | $\mathcal{V}_2 \leftarrow \mathcal{V}_{2,tmp}$;
11 | **end**
12 **end**
13 **return** $\mathcal{V}_1, \mathcal{V}_2$

$MinimumCutPhase(\mathcal{V}, A_w, a)$:

14 $\mathcal{V}_{tmp} \leftarrow \{a\}$;
15 **while** $\mathcal{V}_{tmp} \neq \mathcal{V}$ **do**
16 | find the most tightly connected vertex v; `// see 3.11`
17 | $\mathcal{V}_{tmp} \leftarrow \mathcal{V}_{tmp} \cup \{v\}$;
18 **end**
19 $s \leftarrow$ second last added vertex;
20 $t \leftarrow$ last added vertex;
21 $\mathcal{V}_1 \leftarrow \mathcal{V}/\{t\}$; `// first part`
22 $\mathcal{V}_2 \leftarrow \{t\}$; `// second part`
23 $cost \leftarrow$ calculate cost; `// see 3.12`
24 $\mathcal{V} \leftarrow \mathcal{V}/\{t\}$;
25 $A_w \leftarrow$ shrink A_w by merging the s- and t-vertex;
26 **return** $\mathcal{V}, \mathcal{V}_1, \mathcal{V}_2, cost$

In the *MinimumCut*, *MinimumCutPhase* will be called iteratively until all vertices are merged. The cut-of-the-phase with the minimum cost, recorded by lines 7 to 11, is the desired minimum cut and will be returned (line 13).

3.4. Assignment of Priorities

The idea of prioritizing agents was firstly proposed in [EL87], but without using any criteria to assign priorities, i.e., priorities were assigned arbitrarily. Sequential trajectory planning has been used by many researchers [ČNKS15, VSS10, CNS+13]. Recall that the problem of incompleteness of sequential trajectory planning is that some trajectory planning orders result in infeasibilities of some agents while there exists at least one trajectory planning order allowing all agents find feasible trajectories. There are mainly two strategies handling this problem. The first is to try to avoid this problem by using suitable criteria to assign priorities. The second is to reduce the possibility of infeasibilities once the priorities have already been assigned.

Priority Assignment Criteria

Priority assignment criteria have been proposed in recent years to avoid the problem of incompleteness in sequential trajectory planning. For example, in [Buc89], a heuristic strategy assigning priorities was proposed. Consider a pair of agents. A higher priority will be given to an agent if its straight-line motion from its start point to the goal point intersects with the goal point of another agent or if its start point intersects with the straight-line motion of another agent. For the case where two agents' straight-line motions intersect at their middle, the author did not mention how to assign priorities but randomly gave a higher priority to one agent and adapted another agent's speed to avoid collisions. In Fig. 3.1, vehicle 1 moves to the right, and vehicle 2 turns left. If vehicle 2 is "unluckily" assigned a higher priority, vehicle 1 would need to stop and wait for a long time until vehicle 2 has passed. In this case, however, it is obviously better to assign a higher priority to vehicle 2.

In [LCS16], the authors assigned priorities based on the so-called future collision assessment (FCA). For each agent, its FCA is computed by counting the number of collisions of their reference trajectories with either the actual trajectories of agents if they have already finished planning or the reference trajectories of agents if they are waiting for planning. At each time step, the agent with the highest FCA is assigned the highest priority and thus plans its trajectory first; next, the FCA of all the remaining agents will be computed again, and the one with the highest FCA will be assigned the second highest priority. This is iterated until the last agent. This strategy aims to let vehicles with more constraints on their trajectory optimization problems plan trajectories first, as they have higher possibilities of failing to find feasible trajectories. This strategy may avoid infeasibilities effectively but lacks the consideration of traffic performance. Let us look at Fig. 3.1 again. Vehicle 2 has four collision points with others on its reference path (with vehicles $\{1, 3, 4, 6\}$), while vehicle 1 has only three collision points with others (with vehicles $\{2, 3, 4\}$, note that if two vehicles have the same reference path, for example, vehicles 1 and 3, only one

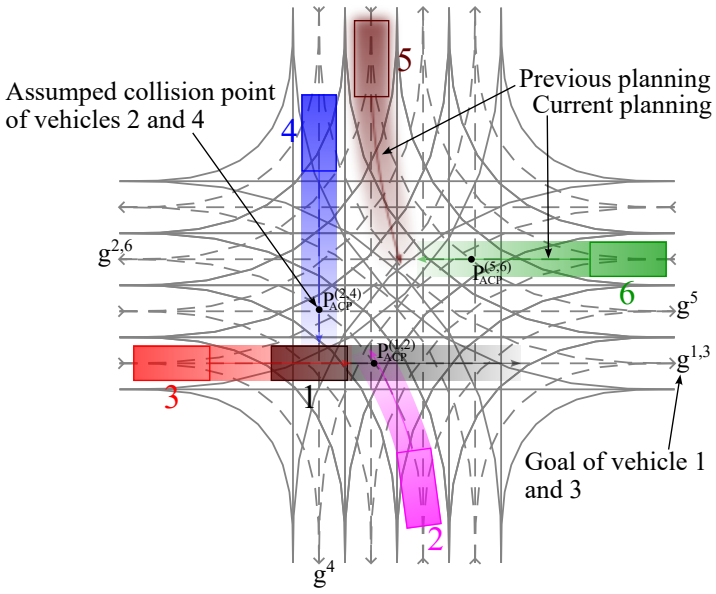


Figure 3.1.: Vehicles at a four-legged intersection with two lanes in each direction. Predicted trajectories are shown in arrows.

collision point is counted). Therefore, a higher priority will be assigned to vehicle 2, which leads to an unnecessarily long waiting time by vehicle 1 for vehicle 2. In addition, for the case where agents have the same number of collision points on their reference paths, the one that is (randomly) selected first will have a higher priority. If we remove all other vehicles except vehicles 1 and 2 in Fig. 3.1, both vehicles will have exactly one collision point on their reference paths. Therefore, vehicle 1 may still have a lower priority than vehicle 2.

In [dFS13, KSM+19], the authors assigned priorities based on the so-called time-to-react, that is, the maximum duration that one agent can keep moving until it must change its maneuvers to avoid collisions or avoid entering the intersection. Higher priorities were assigned to agents with lower time-to-react. In [LRX21], the authors grouped agents based on their reference trajectories and assigned priorities to groups, with agents inside each group having the same priority. In concrete, at each time step, three agents could form a triple if their reference trajectories at this time step were close enough. All triples were stored in a list, and each of them is called an element in the list. Next, the element that appeared most often in the list would be selected and assigned the highest priority. After this, the involved agents would be removed from the list to avoid one agent being selected multiple times. This was repeated until the list became empty. After that, each group of agents planned

trajectories in sequence. However, inside each group, they planned in a centralized way, which is time-consuming compared with distributed trajectory planning.

Reduce the Possibility of Infeasibilities

To reduce the infeasibility possibility once the priorities have been assigned, the authors of [ČNKS15] let agents with higher priorities avoid the start points of agents with lower priorities while planning. The authors in [KRSH07] forced agents with higher priorities to avoid the one-step-shifted previous trajectory of agents with lower priorities. It was proved that if feasible trajectories exist at the initial time step, they will always be found in the future. However, this strategy may cause deadlocks in some scenarios. Deadlocks are situations where vehicles mutually block each other [ČNKS15]. In Fig. 3.1, vehicle 6 has a higher priority than vehicle 5. The previously predicted trajectory of vehicle 5 is shown in a blurred blue arrow, and the footprint is shown in a blurred blue area. All other arrows and colored areas are the trajectories predicted at the current time step and the corresponding footprints. As we can see, vehicle 6 brakes slightly to avoid the previous trajectory of vehicle 5 and sends this plan to vehicle 5. Vehicle 5, however, is blocked by vehicle 6's predicted trajectory and unable to pass in front of it; otherwise, vehicle 5 will hit its right lanelet boundary. After some time, as expected, both vehicles will stop in front of each other, i.e., a deadlock occurs.

3.5. Avoidance of Deadlocks

Deadlocks are situations where agents could find feasible trajectories avoiding collisions but mutually block each other, e.g., agents stop in front of each other and do not collide. One should pay attention to deadlocks while avoiding collisions.

In [CES71], the authors modeled the occupied sets of vehicles as resources. Deadlocks occur when some other vehicles are competing for those resources before they are released. A notable work in [KSM+19] also uses the concept of resource, e.g., the collision regions are considered shared resources that can only be accessed by one vehicle at a time. In this work, an MPC-based strategy was proposed to avoid collisions and deadlocks at intersections. Vehicles are only allowed to enter an intersection if their predicted trajectories cover the whole intersection in the prediction horizon, i.e., they have to wait if they cannot pass the intersection in the prediction horizon. This strategy is effective in avoiding deadlocks at the intersection as vehicles will either pass the intersection without stop or wait in front of it. However, the required number of prediction horizon must be at least so high that vehicles could be able to plan a trajectory to pass the whole intersection. The larger the intersection,

the higher the needed number of prediction horizon. This is problematic because a higher prediction horizon increases the trajectory planning time and impairs the real-time capacity of the proposed strategy in larger intersections.

The Banker's algorithm has been widely used in traffic control. Edsger Wybe Dijkstra developed the Banker's algorithm for computer operating systems to handle resource allocation problems. This algorithm is conservative as in operation systems, only very little information about future resource requirements of processes is known [LRF98]. However, this is not the case for NCSs where, for example, the destinations or reference paths of vehicles can be predefined. Some authors prevented deadlocks use the modified Banker's algorithm [SB94, BB08, KPBB11]. Some works modeled the traffic road by Petri Net – a class of directed graphs, whose nodes represent lanelet intersections or stations where vehicles can load/unload and whose edges represent lanelets. In [WZ07], the authors prevented deadlocks by avoiding conflict cycle chains. A drawback is the inconsideration of rerouting. If, for example, a vehicle is damaged at a lanelet, all other vehicles routed through this lane could no longer arrive at their destinations.

Many works in cooperative intersection management (CIM) focus on preventing deadlocks at intersections, where center controllers manage the order according to which vehicles should cross intersections. In [BF17, BJF18], the authors proposed an intersection management system for platoons. In another notable work [PBL+19], a network of intersections is divided into four zones: entrance, conflict, storage, and exit zones. The entrance (exist) zones are for entering (exiting) the road network. The conflict zones consist of intersections, while the storage zones are lanes connecting the intersections. Each lane is divided into multiple small elementary cells such that each of them cannot be occupied by more than one vehicle. A protocol is proposed to prevent collisions and deadlocks. Collisions are avoided by ensuring that each elementary road cell is occupied by only one vehicle at a time. Deadlocks are avoided by ensuring that vehicles are only unlocked if the following three conditions are satisfied: 1) their precedent vehicles occupying the same zone are unlocked, 2) others do not reserve all precedent road cells, and 3) the number of unlocked vehicles do not exceed its lane capacity. For comprehensive surveys on CIM, readers are referred to [CE16, GS22].

4. A Framework Combining Sequential and Parallel Trajectory Planning

This chapter presents a framework combining sequential and parallel trajectory planning for NCSs.

An overview of the proposed framework is given in Fig. 4.1. Here, reachability analysis is used to calculate the reachable set $\mathcal{R}^{(\square)}$ of each vehicle $\square = 1, \ldots, N$, which will be used in trajectory planning to address the prediction inconsistency problem between parallel planning vehicles (Section 4.1). Vehicles will be divided into groups to alleviate the conservativeness of this approach. Loosely speaking, vehicles in the same group plan trajectories sequentially, and in different groups plan in parallel. This is not strictly true because uncoupled vehicles in the same group could plan in parallel without risking the prediction inconsistency problem, as suggested in [Alr17, Sec. 3.5]. As demonstrated in Section 4.1.4, highly coupled vehicles – that is, vehicles more likely to collide – should be in the same group. Section 4.2 and Section 4.3 are served to solve this vehicle grouping problem. Section 4.2 proposes an algorithm to determine the edge-weighted DAG modeling the coupling situation of vehicles, where the calculated reachable sets will be used to formally determine whether two vehicles are coupled. Highly coupled vehicles are assigned high coupling weights. Based on the weighting matrix A_w of the DAG, Section 4.3 presents a vehicle grouping algorithm, which restricts the number of computation levels of each group by a user-defined parameter n_{CLs}. Next, a strategy improving feasibility, dealing with infeasibilities, and preventing deadlocks is proposed in Section 4.4.1, Section 4.4.2, and Section 4.4.3, respectively. Finally, the communications between vehicles are introduced in Section 4.5, and the overall algorithm of the proposed framework is summarized in Section 4.6.

4.1. Dealing with the Prediction Inconsistency Problem Between Parallel Planning Vehicles

The prediction inconsistency problem will be addressed by letting lower-priority vehicles consider the reachable sets of their coupled vehicles with higher priorities. Recall the example illustrating the risk of this problem provided in Fig. 1.1(b), which is shown again in Fig. 4.2(a). Here, vehicle 2, with a lower priority, assumes

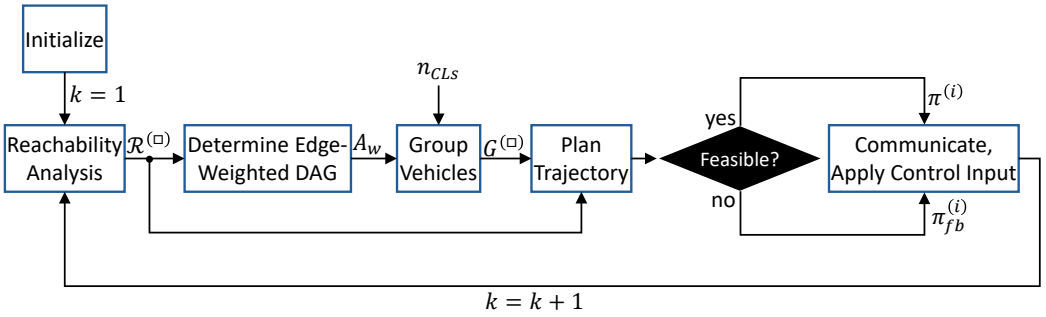
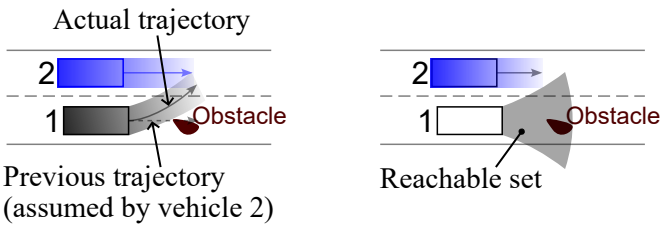


Figure 4.1.: Framework overview. Some explanations: k: time step. $\mathcal{R}^{(\square)}/G^{(\square)}$: reachable set/group of vehicle $\square$, with $\square = 1, \ldots, N$.

that vehicle 1 will keep its previous plan, which deviates from its actual trajectory. Therefore, vehicle 2 endangers vehicle 1 as it plans a trajectory intersecting with that of vehicle 1. If vehicle 2 plans a trajectory avoiding the reachable set instead of an assumed trajectory of vehicle 1, it will safely not cause any collision risks to vehicle 1. As shown in Fig. 4.2, vehicle 2 brakes to avoid vehicle 1's reachable set.



(a) Consider previous trajectory. (b) Consider reachable set.

Figure 4.2.: Address the prediction inconsistency problem.

The following three sections propose a fast method to calculate the reachable sets of nonlinear systems while not linearizing system dynamics. Thanks to the kinematic single-track model's invariance property, most of the computation load can be divided into offline calculations, where offline reachable sets will be computed and stored in a database. During online operating, the actual reachable sets are determined by simply translating and rotating the offline calculated reachable sets according to the current poses of vehicles. Since the offline reachable sets do not contain the position information of vehicles, they will also be called local reachable sets. Correspondingly, the actual reachable sets will be named global reachable sets.

Firstly, Section 4.1.1 explains how reachable sets are represented in this thesis. Section 4.1.2 proposes two algorithms to calculate the local reachable sets of nonlinear dynamic systems. Section 4.1.3 describes the calculation of the global reachable sets

based on the offline calculated local reachable sets and provides a detailed example. Section 4.1.4 remarks on the drawback of the proposed approach dealing with the prediction inconsistency problem of parallel trajectory planning and proposes a solution to mitigate it.

4.1.1. Set Representation

Polygons defined by 2-D vertices are used to represent sets. The proposed framework is implemented in MATLAB. MATLAB provides a class called *polyshape*, which can be utilized to store polygons. Each object of this class is used to represent a reachable set. The intersection, union, and difference of sets are computed using the MATLAB built-in function $intersect : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^2$, $union : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^2$, and $subtract : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^2$. Besides, one can use $overlaps : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{B}$ to check whether two sets intersect if the intersecting area is not of interest, where $\mathbb{B}$ denotes the boolean domain. In addition, the function $boundary : polyshape \to \mathbb{R}^2$ queries the coordinates of all vertices of a *polyshape*.

4.1.2. Calculation of Local Reachable Sets

The local reachable set of a system is calculated based on the MPA discretizing its states. The resulting reachable set's accuracy depends on the fidelity of the MPA. The proposed algorithm does not use any approximation and thus does not introduce any additional errors.

Brute-force Algorithm

Under the MPC framework, reachable sets of a system at each time step $k \leq H_p$ within the prediction horizon are of interest. For this purpose, a brute-force algorithm, Algorithm 2, is given. The first for-loop (lines 1 to 5) initializes n_T one-node trees with each root node storing a distinct trim primitive, where n_T is the number of trim primitives. Each node n in the trees stores four pieces of information: its trim primitive, x- and y-coordinates, and yaw angle. The operator $n(\square) : \{trims, x, y, yaw\} \to \mathbb{R}$ returns the information of $\square$. For example, $n(trim)$ queries the trim of node n while $n(x)$ queries its x-coordinate. The function $addNode : T \times \mathbb{N} \times \mathbb{N} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \to T$ (see line 4 and 17) adds a new node to a tree. It has six arguments. The first is the target tree, and the second specifies the level to which the node should be added, and the remaining four correspond to the information stored in the added node. Recall that the level of a root node is 0. After each time step, the trees will grow by one level. After n_T trees have been initialized, the second for-loop (lines 6 to 24) iterates all steps in the prediction horizon, while the third for-loop (lines 7 to 23) iterates all trees. At the beginning of the third

Algorithm 2: Calculate local reachable set (brute-force algorithm)

Input: trim primitives T_p with n_T trims, motion primitives M_p, prediction horizon H_p

Output: each local reachable set of trim i at time step k: $\mathcal{R}_{local,i}^{(k)}$, $\forall k \in \{1, \ldots, H_p\}, \forall i \in \{1, \ldots, n_T\}$

1 **for** $i \in 1, \ldots, n_T$ **do**
2 $T_i \leftarrow$ initialize an empty tree;
3 $t \leftarrow T_p(i)$; `// get the trim primitive with index` i
4 $T_i \leftarrow addNode(T_i, level = 0, trim = t, x = 0, y = 0, yaw = 0)$; `// add a new node with trim` t `to level` 0 `of tree` T_i
5 **end**
6 **for** $k = 1, \ldots, H_p$ **do**
7 **for** $T_i \in \{T_1, \ldots, T_{n_T}\}$ **do**
8 $tMPs \leftarrow \{\}$; `// store all translated motion primitives`
9 $parentNodes \leftarrow T_i(k-1)$; `// nodes of level` $k-1$ `of tree` T_i
10 **forall** $n_p \in parentNodes$ **do**
11 $t_p \leftarrow n_p(trim)$; `// get the trim of node` n_p
12 $childTrims \leftarrow$ get all the switchable trims of trim t_p;
13 **forall** $t_c \in childTrims$ **do**
14 $m \leftarrow M_p(t_p, t_c)$; `// motion primitive connecting trim` t_p `and` t_c
15 $m_t \leftarrow$ translate m to $[n_p(x), n_p(y)]$ and rotate it by $n_p(yaw)$;
16 $x_e, y_e, yaw_e \leftarrow$ get the x-, y-coordinate, and yaw angle of the end point of m_t;
17 $T_i \leftarrow addNode(T_i, k, t_c, x_e, y_e, yaw_e)$;
18 $tMPs \leftarrow tMPs \cup m_t$;
19 **end**
20 **end**
21 $OSs \leftarrow \mathcal{O}(tMPs)$; `// occupied set of each element in` $tMPs$
22 $\mathcal{R}_{local,i}^{(k)} \leftarrow$ union all the occupied sets in OSs;
23 **end**
24 **end**
25 **return** $\mathcal{R}_{local,i}^{(k)}, \forall k \in \{1, \ldots, H_p\}, \forall i \in \{1, \ldots, n_T\}$

for-loop, a variable is created (line 8) to store all motion primitives translated by line 15. In essence, the reachable set of a trim in time step k is calculated by the union of all the occupied sets of motion primitives translated in this time step. Those motion primitives correspond to edges connecting nodes in (the previous) level

$k-1$ and nodes in (the current) level k. The nodes in the previous level are called parent nodes while the nodes in the current level are their child nodes. Similarly, the trims stored in them are called parent trims and child trims. Child trims must be switchable from their parent trims (line 12). Recall that trim i is switchable from trim j if a directed edge from trim j to trim i exists in the directed graph representing the corresponding trim primitives. After translating the target motion primitive (line 15), the x-, y-coordinates, and yaw angle of the end point of the translated motion primitive will be stored in the child node and added to level k of tree T_i (line 17). Finally, the occupied set of each translated motion primitive is calculated using operator $\mathcal{O}()$ (see Definition 14 in Section 2.3) in line 21, and the (local) reachable set of trim i at time step k is the union of the resulting occupied sets (line 22).

Dynamic Programming

Although the local reachable sets are calculated offline, the computation time of Algorithm 2 is impractical if a large prediction horizon is used because the number of expanded nodes will grow exponentially. This problem is addressed using dynamic programming. More specifically, the local reachable sets of the first half prediction horizon are calculated by Algorithm 2, while that of the second half prediction horizon is calculated efficiently based on the reachable sets of the half prediction horizon and the created trees. The pseudocode is presented in Algorithm 3.

In Algorithm 3, firstly, the number of the half prediction horizon $H_{p,half}$ is calculated (line 1). Note that rounding up is needed if H_p is an odd number. Next, line 2 calls Algorithm 2 to calculate the local reachable sets of the first half prediction ($k = 1, \ldots, H_{p,half}$). Line 3 is needed if the recursive feasibility is guaranteed by forcing the last trim primitive to be an equilibrium trim. In this case, Algorithm 2 will add this constraint on the trim primitive of nodes on the last level of the trees. The resulting reachable sets at the last prediction horizon will be called constrained reachable sets. However, the recursive feasibility must not be guaranteed here, as only half of the prediction horizon should be computed by Algorithm 2. The constrained reachable sets are still useful because they can be utilized to calculate the reachable sets at the last time step of the prediction horizon. The full reachable set at time step $H_{p,half}$ can be computed based on the nodes of level $H_{p,half}$. This can be done by lines 6 to 24 in Algorithm 2 with k being a constant value, $H_{p,half}$. The newly expanded nodes will replace the old nodes on level $H_{p,half}$ (see line 3 in Algorithm 3). Finally, lines 4 to 15 calculate the reachable sets of the second half prediction horizon ($k = H_{p,half} + 1, \ldots, H_p$). Line 6 initializes a variable to store all the translated reachable sets. Here, the reachable set of trim i at time step $H_{p,half}$ will be translated to the nodes located at level $k - H_{p,half}$ of tree T_i, called basis

nodes. After translating (line 9), the resulting reachable set will be stored (line 10). The local reachable set of trim i at time step $k(> H_{p,half})$ is the union of all the translated and stored reachable sets (line 12).

Algorithm 3: Calculate local reachable set (dynamic programming)

Input: trim primitives T_p with n_T trims, motion primitives M_p, prediction horizon H_p

Output: each local reachable set of trim i at time step k: $\mathcal{R}^{(k)}_{local,i}$, $\forall k \in \{1,\dots,H_p\}, \forall i \in \{1,\dots,n_T\}$

1 $H_{p,half} \leftarrow H_p/2$; `// round up if` H_p `is not an even number`
2 Call Algorithm 2 with half of the prediction horizon $H_{p,half}$;
3 Compute the reachable set at time step $H_{p,half}$ without guaranteeing the recursive feasibility, and replace the nodes in level $H_{p,half}$ with the newly expanded nodes;
`// Calculate the second half prediction horizon:`
4 **for** $k = H_{p,half}+1,\dots,H_p$ **do**
5 **for** $T_i \in \{T_1,\dots,T_{n_T}\}$ **do**
6 $tRSs \leftarrow \{\}$; `// store all translated reachable sets`
7 $basisNodes \leftarrow T_i(k - H_{p,half})$; `// nodes of level` $k - H_{p,half}$ `of tree` T_i
8 **forall** $n_b \in basisNodes$ **do**
9 $r^t \leftarrow$ translate $\mathcal{R}^{(H_{p,half})}_{local,i}$ to $[n_b(x), n_b(y)]$ and rotate it by $n_b(yaw)$;
10 $tRSs \leftarrow tRSs \cup r^t$;
11 **end**
12 $\mathcal{R}^{(k)}_{local,i} \leftarrow$ union all the translated reachable sets in $tRSs$;
13 **end**
14 **end**
15 **return** $\mathcal{R}^{(k)}_{local,i}, \forall k \in \{1,\dots,H_p\}, \forall i \in \{1,\dots,n_T\}$

Showcase Example Calculating Local Reachable Sets

Fig. 4.3 visualizes a part of the execution process of Algorithm 3 for a system with simple trim primitives shown in Fig. 4.4 (note that the edge from each trim to itself is not depicted). More concrete, the calculation process of the local reachable sets of trim 1 (the equilibrium trim) is visualized in Fig. 4.3. Consider a prediction horizon H_p of 4. The local reachable sets of the first two time steps are calculated by Algorithm 2 (the brute-force algorithm), where the expanded nodes are shown as opaque dots with different colors. Each node's color indicates its trim primitive

while its position in the figure corresponds to the x- and y-coordinates stored in it. The local reachable sets of the last two time steps are calculated efficiently using Algorithm 3, where no node is expanded. In comparison, the nodes expanded by the brute-force algorithm at the last two time steps are depicted as transparent dots. One may notice the enormous difference between the number of opaque and transparent dots. This is why the total computation time is reduced largely by Algorithm 3. The effectiveness of this algorithm will be further evaluated in Section. 5.1. The reachable set of the first, second, third, and last time step is depicted in solid, dashed, dash-dotted, and dotted polygons, respectively. Note that the head of each reachable set is unsmooth because the system inputs are discretized by trim primitives. Smoother reachable sets are expected if higher-fidelity trim primitives are used.

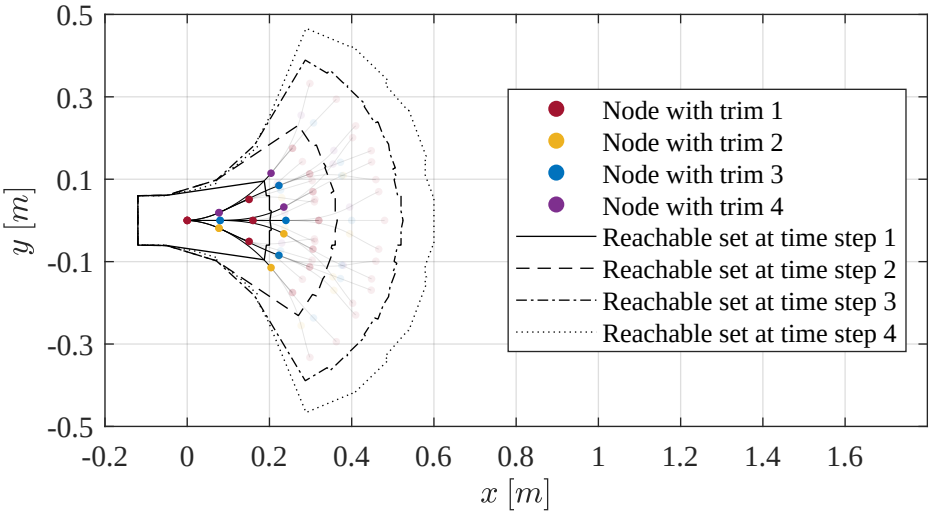


Figure 4.3.: Visualization of the calculation of the local reachable sets of trim primitive 1 in Fig. 4.4.

4.1.3. Calculation of Global Reachable Sets

In the previous section, the local reachable sets of each trim primitive i at each time step k are calculated. This section details how to determine the global reachable sets, which consider a vehicle's current position and yaw angle, efficiently based on the offline local reachable sets when operating online.

The principle is similar to the usage of the offline motion primitives (see Fig. 2.4). The global reachable sets are obtained by translating the local reachable sets to the

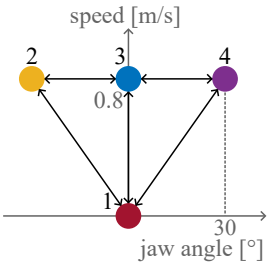


Figure 4.4.: Simple trim primitives.

current position of the vehicle and rotating them by the vehicle's current yaw angle. Fig. 4.5 illustrates this principle. The local reachable sets of trim 1 determined in Fig. 4.3 are depicted as transparent on the left side. Consider a vehicle with yaw angle $\psi = 30°$ at the position $x = 1, y = -0.4\,m$, and its states are at trim 1. Its reachable sets are determined by firstly translating the local reachable sets to the point $x = 1, y = -0.4\,m$ and then rotating them by $\psi = 30°$ counterclockwise. The results are depicted in opaque polygons on the right side of Fig. 4.3. Note that the translation and rotation computation time is almost negligible, as only a couple of matrix multiplications are performed.

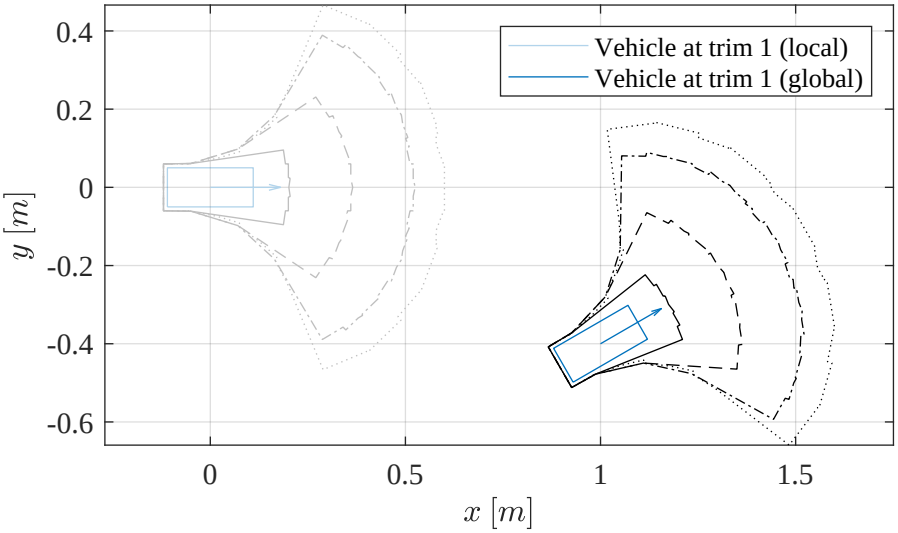
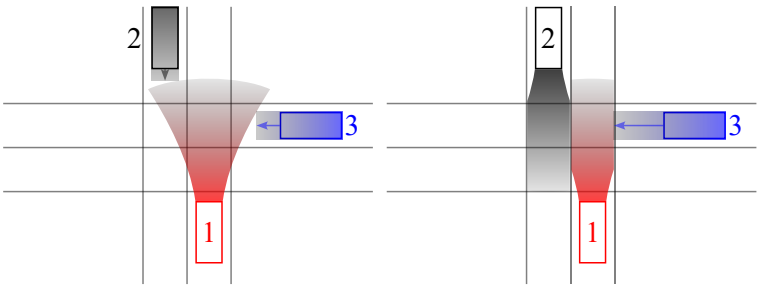


Figure 4.5.: Calculation of global reachable sets based on the offline local reachable sets.

Bound Reachable Sets

As vehicles are assumed to always stay in their lanelet boundaries, one can bound its reachable set using its lanelet boundaries, which the MATLAB function *intersect* can do. The advantages are twofold. First, it alleviates the conservativeness of the reachability analysis since one vehicle's reachable set does not span in space as much as before. In Fig. 4.6, vehicle 1 has the highest priority while vehicle 3 has the lowest. After bounding the reachable set of vehicle 1, vehicle 3 plans a less conservative trajectory. Second, vehicles will not be considered coupled if their lanelet boundaries do not intersect, avoiding thus unnecessary couplings. Before bounding reachable sets, vehicles 1 and 2 are coupled since their reachable sets intersect (see Definition 16). This coupling does not exist if their reachable sets are bounded (see Fig. 4.6(b)). Therefore, vehicle 2 can plan its trajectory without considering vehicle 1.



(a) Unbounded reachable set. (b) Bounded reachable set.

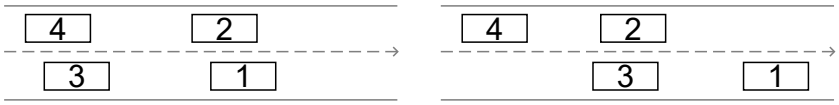
Figure 4.6.: Bound reachable sets using lanelet boundaries. (a) Vehicle 3 plans a conservative trajectory. (b) Vehicle 3's trajectory becomes less conservative.

4.1.4. Drawback of Consideration of Reachable Sets

Until now, the prediction inconsistency problem can be addressed by reachability analysis. However, the conservativeness of this approach is obvious. For example, in Fig. 4.2, if there is no obstacle in front of vehicle 1, vehicle 2 will not need to brake since vehicle 1 will keep its speed going straight.

As one may notice, parallel trajectory planning becomes conservative if its prediction inconsistency problem is addressed by reachability analysis. Considering this, one can use sequential trajectory planning in parallel trajectory planning if the total trajectory planning time allows, i.e., let some vehicles plan in sequence. If vehicles are considered in a group if they plan trajectories in sequence. A question that arises is: *Which vehicles should be in the same group?* In Fig. 4.7(a), four vehicles move rightwards and plan trajectories in parallel. Assume that their trajectory planning time is equal, t_0,

leading to a total trajectory planning time of t_0 ($T_{ct,traj} = t_0$). If $T_{ct,traj}$ is allowed to be $2t_0$, one can let vehicles 1 and 2, vehicles 3 and 4 plan sequentially, as they are close to each other, i.e., two groups can be formed: $G_1 = \{1, 2\}, G_2 = \{3, 4\}$. This reduces the total trajectory planning time to $2t_0$ while risking the prediction inconsistency problem between the following vehicle pairs: $\{1, 3\}, \{1, 4\}, \{2, 3\}$, and $\{2, 3\}$. This risk is high if two closely moving vehicles plan trajectories in parallel (see Fig 4.2(a)). In Fig 4.7(b), one may form the following three groups: $G_1 = \{1\}, G_2 = \{2, 3\}, G_3 = \{4\}$, which also leads to $T_{ct,traj} = 2t_0$. If $T_{ct,traj}$ is allowed to be longer, $3t_0$, it is hard to decide which vehicles should form groups in Fig 4.2(a). For example, should it be $G_1 = \{1, 2, 3\}, G_2 = \{4\}$? Or should it be $G_1 = \{1\}, G_2 = \{2, 3, 4\}$? Or is it better to keep the previous option which results in a total trajectory planning time of $2t_0$? In Fig 4.2(b), one may have $G_1 = \{1, 2, 3\}, G_2 = \{4\}$. Similarly, one can also have $G_1 = \{1\}, G_2 = \{2, 3, 4\}$. Both options result in $T_{ct,traj} = 3t_0$. As one may see, it is nontrivial to form parallel groups, especially at intersections where vehicles move in different directions.



(a) Scenario 1. (b) Scenario 2.

Figure 4.7.: Different vehicle grouping options. (a) Vehicles 1 and 2, vehicles 3 and 4 are close to each other. (b) Vehicles 2 and 3 are close to each other.

In- and Cross-Group Couplings

Next, it will be answered why it is better to put highly coupled vehicles in the same group.

Each vehicle may have couplings with vehicles in the same group as well as in different groups. The former is called in-group couplings while the latter is cross-group couplings. In-group couplings are considered by letting vehicles with lower priorities avoid the predicted occupied sets of coupled vehicles with higher priorities when trajectory planning. In comparison, cross-group couplings are considered by avoiding either the predicted occupied sets or reachable sets of coupled vehicles with higher priorities.

Proposition 1. *Compared with in-group couplings, the consideration of cross-group couplings leads to more conservative trajectories of lower-priority vehicles or higher possibilities that they cannot find feasible trajectories. In the best case (see case 1 below), the difficulty of avoiding those two types of couplings is the same.*

Proof. Consider arbitrary two coupled vehicles i and j and at an arbitrary time step k. Assume vehicle i has a higher priority without loss of generality.

- If they are in the same group, vehicle i plans its trajectory $\pi^{(i)}_{([k,k+H_p]|k)}$ (for H_p time steps) and sends the predicted occupied sets $\mathcal{O}(\pi^{(i)}_{([k,k+H_p]|k)})$ to vehicle j. Vehicle j receives and avoids it while planning its trajectory $\pi^{(j)}_{([k,k+H_p]|k)}$.
- If they are in different groups, the vehicle with the highest priority in each group plan trajectory first. There are two cases:
  - Case 1: Vehicle j starts planning after vehicle i has finished its planning.
  - Case 2: Vehicle j starts planning before vehicle i has finished its planning.

For case 1, vehicle j can receive the predicted occupied sets of vehicle i, i.e., $\mathcal{O}(\pi^{(i)}_{([k,k+H_p]|k)})$; therefore, its planned trajectory $\pi^{(j)}_{([k,k+H_p]|k)}$ will be the same as if they are in the same group. For case 2, however, vehicle j has to calculate and avoid the reachable sets of vehicle i, i.e., $\mathcal{R}^{(i)}_{([k,k+H_p]|k)}$. According to Definition 15, a reachable set considers all the possible trajectories of a certain time; thus, it holds that $\mathcal{O}(\pi^{(i)}_{([k,k+H_p]|k)}) \subset \mathcal{R}^{(i)}_{([k,k+H_p]|k)}$. Therefore, vehicle j in case 2 must avoid larger sets while trajectory planning, leading to a more conservative trajectory or a higher possibility that it cannot find a feasible trajectory. □

Due to Proposition 1 and based on the fact that the collisions between highly coupled vehicles are generally hard to avoid, they should be put in the same group planning trajectories in sequence, and their couplings can therefore be avoided by considering the predicted occupied sets instead of the reachable sets of higher-priority vehicles. The next section, Section 4.2, proposes an algorithm to determine the edge-weighted DAG of NCSs. The edge weights indicate the coupling degrees between vehicles, based on which vehicles can be grouped in Section 4.3.

4.2. Determination of Edge-Weighted Directed Acyclic Graph

This section proposes an algorithm to determine the edge-weighted DAG modeling the coupling situation of NCSs. Section 4.2.1 describes how the road is modeled in this thesis. Section 4.2.2 introduces the idea of the shortest time to achieve a collision (STAC), which will be used in Section 4.2.3 to calculate the coupling weights between vehicles. Section 4.2.4 summarizes the algorithm. Section 4.2.5 gives suggestions on dealing with coupling cycles, which may lead to deadlocks. Finally, Section 4.2.6 explains how to assign priorities for each group of vehicles.

Recall Definition 16 for the term *coupled* in Section 2.4.1. Since MPC is used, this thesis declares that two vehicles are coupled if they may collide in the prediction

horizon (in H_p time steps). This can be formally checked by reachability analysis, i.e., two vehicles are coupled if their reachable sets intersect in H_p time steps. The function $isCoupled(i, j, k) : \mathcal{V} \times \mathcal{V} \times \mathbb{N} \rightarrow \mathbb{B}$ returns true if two vehicles i and j are coupled at time step k.

$$\begin{aligned} &isCoupled(i, j, k) = true \iff \exists h \in 0, ..., H_p - 1 : \\ &\mathcal{R}^{(i)}_{([k+h,k+h+1]|k)} \cap \mathcal{R}^{(j)}_{([k+h,k+h+1]|k)} \neq \emptyset. \end{aligned} \tag{4.13}$$

Note that each coupling is mutual, i.e., $isCoupled(i, j, k) \iff isCoupled(j, i, k)$.

4.2.1. Modeling of Road

This thesis uses a road map consisting of lanelets for autonomous driving.

Definition 18 (Lanelet [BZS14])**.** *Lanelet describes an atomic lane segment characterized by its left and right bound. The bounds are polylines and allow for an arbitrary precise approximation of lane geometries.*

In addition, the centerlines of lanelets are used in this thesis, which are calculated by taking the mean value of the polylines presenting the left and right bounds. The start point and end point of a lanelet are defined as the first point and last point of its centerline. A visualization is given in Fig. 4.8. In [BZS14, AM16], the definitions of adjacent lanelets are purely based on their spatial relation, e.g., two lanelets are longitudinal-adjacent if one lanelet's end point is identical with one lanelet's start point. However, two vehicles could be coupled even if they are at two lanelets with no common point (see lanelets 1 and 4 in Fig 4.8). Here, adjacent lanelets are redefined:

Definition 19 (Adjacent lanelets)**.** *The lanelets of two coupled vehicles are adjacent (see Definition 16).*

Totally, seven kinds of adjacent types fulfill Definition 19:

- Same: Two vehicles could be coupled if they are at the same lanelet.
- Left-adjacent: A lanelet is left-adjacent to another lanelet if the points of its right bound are identical to the points of the left bound of the other lanelet. In Fig. 4.8, lanelet 4 is left-adjacent to lanelet 3. Mutually, lanelet 3 is right-adjacent to lanelet 4.
- Right-adjacent: Analog to left-adjacent.
- Longitudinal-adjacent: Two lanelets are longitudinal-adjacent if the end point of one lanelet is identical to the start point of the other lanelet. The first lanelet is called the predecessor lanelet, while the second lanelet is called the successor lanelet. For example, in Fig. 4.8, lanelets 3 and 6 are longitudinal-adjacent,

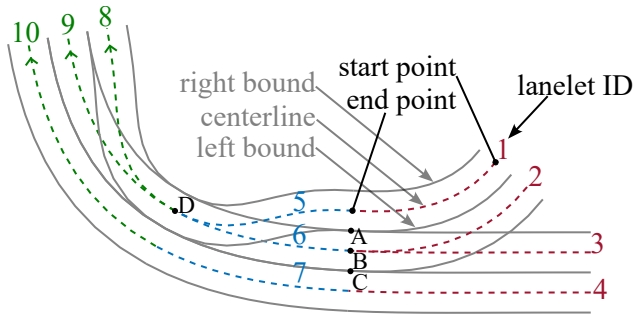


Figure 4.8.: A road segment consisting of ten lanelets. Point A is the merging point of lanelets 1 and 3. Point B is the merging point of lanelets 2 and 3. Point C is the lanelet critical points of lanelets 3 and 4. Point D is the forking point of lanelets 8 and 9.

with lanelet 3 being the predecessor lanelet, and lanelet 6 being the successor lanelet. In addition, the left- or right-adjacent lanelets of longitudinal-adjacent lanelets are also considered longitudinal-adjacent, such as lanelets 4 and 6 in Fig. 4.8.

- Merging-adjacent: Two lanelets have the same end point, called the merging point. In Fig. 4.8, lanelets 2 and 3 are merging-adjacent, with point B being the merging point. The left- or right-adjacent lanelet of a merging lanelet can also be considered the merging lanelet of another merging lanelet. For example, lanelets 1 and 3. Their merging point is point A, that is, the end point of lanelet 1's left bound or lanelet 3's right bound. In addition, the left- or right-adjacent lanelets of two merging lanelets are also considered merging-adjacent, e.g., lanelets 1 and 4. Their merging point is hard to define. One can set it as the middle point of the end points of lanelet 1's left bound and lanelet 4's right bound, which roughly overlaps with point B. Or one can set it simply as point B.
- Forking-adjacent: Two lanelets have the same start point, called the forking point. In Fig. 4.8, lanelets 8 and 9 are forking-adjacent, with point D being the forking point. Similarly, lanelets 8 and 10 will also be considered forking-adjacent lanelets.
- Crossing-adjacent: Two lanelets are crossing. The crossing point cannot simultaneously be the start point or end point of both lanelets. In Fig. 3.1, the lanelet of vehicle 1 and the lanelet of vehicle 2 are crossing-adjacent lanelets, with $P_{ACP}^{(1,2)}$ being the crossing point.

The points mentioned above, such as merging points, forking points, and crossing points, will be referred to as lanelet critical points. The lanelet critical point of two

same lanelets is their end point, and that of longitudinal-adjacent lanelets is the end point of the successor lanelet. For left- and right-adjacent lanelets, their lanelet critical point is the end point of the shared bound.

Based on Definition 19, two arbitrary lanelets could be adjacent if an infinite prediction horizon is used. In this thesis, a moderate prediction horizon is assumed, such that two lanelets can only be adjacent if they are mutually reachable through maximum one longitudinal-adjacent lanelet. For example, lanelets l_i and l_j could be adjacent, $\forall l_i \in \{1, 2, 3, 4\}$ and $\forall l_j \in \{5, 6, 7\}$; but if $l_j \in \{8, 9, 10\}$, they will not be considered adjacent.

Proposition 2. *If two vehicles at two lanelets are not adjacent, they will never collide; therefore, they are not coupled.*

Proof. The proof follows directly from Definition 16 and Definition 19. □

According to [AM16], an adjacent lanelet matrix

$$A_l^R : \mathcal{V}_l \times \mathcal{V}_l \rightarrow \{same, long, left, right, merge, fork, cross, none\} \tag{4.14}$$

stores the relationship of each pair of lanelets, where $\mathcal{V}_l = \{1, \ldots, N_l\}$ denotes a set of lanelet IDs with N_l being the number of lanelets. Note that each lanelet has a distinct ID. $A_l^R(l_i, l_j)$ returns the relationship of lanelets l_i and l_j. The relationship of two lanelets is *none* if they are not adjacent. Since the lanelet critical points will be used in the next section to determine directed couplings between vehicles and their STAC, matrix $A_l^P : \mathcal{V}_l \times \mathcal{V}_l \rightarrow \mathbb{R}^2$ is defined, and $A_l^P(l_i, l_j)$ returns the lanelet critical point of lanelets l_i and l_j. A_l^R will be called adjacent lanelet relationship matrix while A_l^P is named adjacent lanelet critical point matrix.

This thesis expects that the above-mentioned lanelet information will be available in the future for autonomous driving [AM16, ZBS+14].

4.2.2. Determination of Directed Couplings

Firstly, two types of collisions are distinguished:

- Rear-end collision: A rear-end collision occurs when a vehicle hits another vehicle in front of it from behind.
- Side-impact collision: A side-impact collision occurs when a vehicle hits another vehicle from one side.

Next, the assumed collision point (ACP) is defined:

Definition 20 (Assumed collision point)**.** *The ACP is the point at which two coupled vehicles i and j are assumed to collide if they follow their reference paths exactly.*

The following rules are formulated to determine the ACP (denoted by $P_{ACP}^{(i,j)}$) of two coupled vehicles i and j according to the relationship of their reference lanelets:

- Longitudinal- or forking-adjacent: Only rear-end collision is possible. The ACP is the point where the rear vehicle could catch the front vehicle in the shortest time, i.e., it is assumed that the vehicle behind accelerates fully and the vehicle in front takes the emergency brake. In this case, the ACP is also called the assumed overtaking point (AOP), denoted as $P_{AOP}^{(i,j)} \in \mathbb{R}^2$. At longitudinal-adjacent lanelets, the vehicle closer to their lanelet critical point $A_l^P(i,j)$ is the front vehicle and will be assigned a higher priority. If they are at forking lanelets, the opposite holds.
- Same, left-, right-, or merging-adjacent: Both types of collisions are possible. If they move side by side, they would have side-impact collisions, and the ACP is approximately set as the middle point of their centers of gravity. Otherwise, they would have a rear-end collision possibility. In this case, for the same, left-, or right-adjacent lanelets, the ACP is calculated the same as the AOP. For merging-adjacent lanelets, the ACP is their lanelet critical point, i.e., the merging point. In all the four lanelet types, the vehicle closer to the lanelet critical point will be assigned a higher priority.
- Crossing-adjacent: Only side-impact collisions are possible. The ACP is their lanelet critical point, i.e., the crossing point. A higher priority is given to the vehicle that could arrive at the ACP earlier.

A collision type matrix $A_{CT} : \mathcal{V} \times \mathcal{V} \to \{sideImpact, rearEnd\}$ is defined to store the collision types between coupled vehicles. According to the above rules, one can use Equation 4.15 to determine the collision type of each coupled pair of vehicles i and j.

$$A_{CT}(i,j) = \begin{cases} sideImpact & \text{if } A_l^R(l_i,l_j) == cross \vee \\ & \left(A_l^R(l_i,l_j) \in \{same, left, right, merge\} \wedge isSbS(i,j)\right) \\ rearEnd & \text{otherwise} \end{cases} \tag{4.15}$$

Here, the function $isSbS : \mathcal{V} \times \mathcal{V} \to \mathbb{B}$ returns true if vehicles i and j move side by side. Two vehicles are considered to move side by side if and only if the projection of one vehicle's body centerline to another vehicle's body centerline overlaps with another vehicle's body centerline, which must hold mutually for both vehicles. Fig. 4.9 illustrates this principle. In (a), none of the projection overlaps with the other's body centerline. In (b), the projection of vehicle 1's body center line overlaps with vehicle 2's body centerline (the overlapping part is shown in a solid thick line), but this is not the case for vehicle 2. Therefore, they are not considered to move side by side. In (c), both of the projections overlap with each other's body centerline. Thus, they move side by side.

If vehicles i and j have a rear-end collision possibility, their AOP is of interest. Since the exact AOP is hard to calculate, one can calculate it approximately by ignoring the yaw angles, i.e., both vehicles are assumed to move in a straight line. In addition, constant acceleration $a_+(>0)$ and deceleration $a_-(>0)$ are assumed. The point where the front vehicle is overtaken is the approximated AOP. Assume vehicle i is the front and vehicle j is the rear vehicle. Four cases are distinguished when they collide:

1. Vehicle i is still decelerating while vehicle j is still accelerating.
2. Vehicle j is stopped while vehicle j is still accelerating.
3. Vehicle i is stopped while vehicle j has already reached its maximum speed.
4. Vehicle i is still decelerating while vehicle j has already reached its maximum speed.

The time that the rear vehicle needs to catch the front vehicle, i.e., the time to catch (TTC), of case 1 is computed by

$$t_{TTC}^{case1} = -\frac{v_0^{(j)} - v_0^{(i)}}{a_+^{(j)} - a_-^{(i)}} + \sqrt{\left(\frac{v_0^{(j)} - v_0^{(i)}}{a_+^{(j)} - a_-^{(i)}}\right)^2 + \frac{2d_{ist}^{(i,j)}}{a_+^{(j)} - a_-^{(i)}}}, \tag{4.16}$$

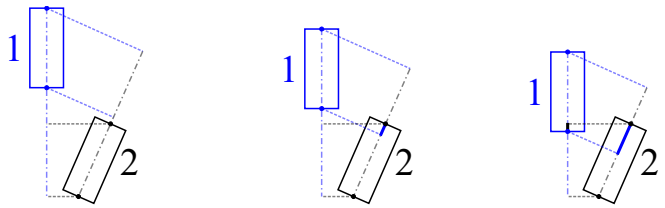
where $d_{ist}^{(i,j)}$ is the distance between vehicles i and j.

Function 4.17 $t_{TTC}(v_0^{(i)}, v_0^{(j)}, a_-^{(i)}, a_+^{(j)}, d_{ist}^{(i,j)}) : \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ is given to calculate the TTC for all the above four cases. The result is denoted as $t_{TTC}^{(i,j)}$.

$$t_{TTC} = \begin{cases} t_{TTC}^{case1} & \text{if } t_-^{(i)} \geq t_{TTA}^{case1} \wedge t_+^{(j)} \geq t_{TTA}^{case1} \\ -\frac{v_0^{(j)}}{a_+^{(j)}} + \sqrt{\left(\frac{v_0^{(j)}}{a_+^{(j)}}\right)^2 + \frac{2d_{ist}^{(i,j)} a_-^{(i)} + {v_0^{(i)}}^2}{a_-^{(i)} a_+^{(j)}}} & \text{if } t_-^{(i)} < t_{TTA}^{case1} \wedge t_+^{(j)} \geq t_{TTA}^{case1} \\ \frac{2d_{ist}^{(i,j)} a_-^{(i)} a_+^{(j)} + a_+^{(j)} {v_0^{(i)}}^2 + a_-^{(i)} \left(v_{max}^{(j)} - v_0^{(j)}\right)^2}{2 v_{max}^{(j)} a_-^{(i)} a_+^{(j)}} & \text{if } t_-^{(i)} < t_{TTA}^{case1} \wedge t_+^{(j)} < t_{TTA}^{case1} \\ \frac{v_{max}^{(j)} - v_0^{(j)}}{a_-^{(i)}} - \sqrt{\left(\frac{v_{max}^{(j)} - v_0^{(j)}}{a_-^{(i)}}\right)^2 - \frac{2d_{ist}^{(i,j)} a_-^{(i)} a_+^{(j)} + a_-^{(i)} \left(v_{max}^{(j)} - v_0^{(j)}\right)^2}{a_-^{(i)} a_+^{(j)}}} & \text{if } t_-^{(i)} \geq t_{TTA}^{case1} \wedge t_+^{(j)} < t_{TTA}^{case1} \end{cases} \tag{4.17}$$

Here, $t_-^{(i)} = v_0^{(i)}/a_-^{(i)}$ and $t_+^{(j)} = v_0^{(j)}/a_+^{(j)}$ is the maximum deceleration and acceleration time of vehicle i and vehicle j, respectively.

If vehicles i and j have a side-impact collision possibility, the time to arrive (TTA) at the ACP is of interest. Similarly to the TTC, the exact TTA is difficult to compute. Therefore, similar assumptions are made: Both vehicles are assumed to take a full



(a) Not side by side. (b) Not side by side. (c) Side by side.

Figure 4.9.: Check if two vehicles move side by side.

(constant) acceleration, and their yaw angles are assumed to always point to the ACP. Function 4.18 $t_{TTA}(v_0, a_+, d_{ist}^{ACP}) : \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ is defined to calculate the approximated TTA in a fast way. $t_{TTA}^{(i,j)}$ and $t_{TTA}^{(j,i)}$ denotes the TTA of vehicle i and vehicle j, respectively.

$$t_{TTA}(v_0, a_+, d_{ist}^{ACP}) = \begin{cases} \frac{\sqrt{v_0^2 + 2a_+ d_{ist}^{ACP}} - v_0}{a_+} & \text{if } \frac{v_{max}^2 - v_0^2}{2a_+} \geq d_{ist}^{ACP} \\ \frac{2a_+ d_{ist}^{ACP} + (v_{max} - v_0)^2}{2a_+ \cdot d_{ist}^{ACP}} & \text{otherwise} \end{cases}, \tag{4.18}$$

where v_0 is the current speed, and a_+ the maximum acceleration of the target vehicle. d_{ist}^{ACP} denotes its distance to the ACP. The if-condition in Equation 4.18 corresponds to cases where it arrives at the ACP while accelerating, while the second condition is for the case where it arrives at the ACP after it has already reached its maximum speed v_{max}. A higher priority is assigned to the vehicle with a lower TTA; that is, the vehicle that could arrive at their assumed collision point earlier has a higher priority.

Next, the STAC $\tau \in \mathbb{R}^+$ can be calculated, an essential parameter characterizing the coupling degree between two coupled vehicles.

Definition 21 (The shortest time to achieve a collision)**.** *The STAC τ is the shortest time that two vehicles could achieve a collision.*

For rear-end collision type, the STAC equals the TTC. Suppose two vehicles have a side-impact collision possibility. In that case, their STAC is the larger one among the TTA of them, as the earlier-arrived vehicle must wait until the second vehicle also arrives and thus collides with it. Equation 4.19 $\tau(i, j) : \mathcal{V} \times \mathcal{V} \to \mathbb{R}^+$ is defined to calculate the STAC of vehicles i and j. The result is denoted as $\tau^{(i,j)}$.

$$\tau(i, j) = \begin{cases} t_{TTC}(i, j) & \text{if } A_{CT}(i, j) == rearEnd \\ \max(t_{TTA}(i, j), t_{TTA}(j, i)) & \text{otherwise} \end{cases} \tag{4.19}$$

4.2.3. Estimation of Coupling Weights

In this subsection, a function is defined to calculate the coupling weight w_e of two coupled vehicles based on their STAC, called STAC-based coupling weight, which serves as an indicator for their coupling degree. As a higher coupling weight corresponds to a stronger coupling degree (see Definition 9), one can conclude that if two vehicles have a higher coupling weight, their collisions will be more difficult to avoid. Note that the coupling weights will only be used to group vehicles. They do not play any direct role in the trajectory planning order or priorities of the vehicles.

STAC-based Coupling Weights

The author declares that if two vehicles can achieve a collision in a shorter time, their coupling is stronger. Therefore, it is rational to let their coupling weight have a negative proportional relationship to their STAC. However, there are cases where two vehicles are coupled stronger while their STAC are the same. Let us look at Fig. 3.1 again. The STAC of vehicles 1 and 2 equals the time vehicle 2 needs to arrive at their ACP $P_{ACP}^{(1,2)}$, i.e., $\tau^{(1,2)} = t_{TTA}^{(2,1)}$. If vehicle 2 could arrive at point $P_{ACP}^{(1,2)}$ earlier than vehicle 3, the STAC of vehicles 2 and 3 would also equal the time that vehicle 2 needs to arrive at the same ACP since vehicles 1 and 3 have the same reference path, i.e., $\tau^{(2,3)} = t_{TTA}^{(2,3)}$. In this case, the coupling weight of vehicles 1 and 2 equals that of vehicles 2 and 3, i.e., $w_e^{(1,2)} = w_e^{(2,3)}$, which is obviously not convincing. Ideally, $w_e^{(2,3)}$ should be smaller than $w_e^{(1,2)}$ since vehicles 2 and 3 tend to collide easier. The so-called waiting time τ_w is introduced to make this possible, that is, the time that the earlier-arrived vehicle needs to wait until the other vehicle also arrives at their ACP. Function 4.20 $\tau_w(i,j) : \mathcal{V} \times \mathcal{V} \to \mathbb{R}^+$ is defined to calculate the waiting time between vehicles i and j, denoted as $\tau_w^{(i,j)}$.

$$\tau_w(i,j) = |t_{TTA}^{(i,j)} - t_{TTA}^{(j,i)}|. \tag{4.20}$$

Since the TTA is only of interest if two vehicles have a side-impact collision possibility, one does not need waiting time in rear-end collisions, i.e., the waiting time in rear-end collisions is always zero.

Formally, a weighting function $w_e(\tau^{(i,j)}, \tau_w^{(i,j)}) : \mathbb{R} \times \mathbb{R} \to (0,1)$ is defined in Equation 4.21 to calculate the coupling weight between two vehicles i and j based on their STAC $\tau^{(i,j)}$ and waiting time $\tau_w^{(i,j)}$, where $(0,1)$ is an open interval from zero to one.

$$w_e(\tau^{(i,j)}, \tau_w^{(i,j)}) = \exp(-\alpha_0 \cdot (\tau^{(i,j)} + \tau_w^{(i,j)})) \tag{4.21}$$

Here, $\alpha_0 \geq 0$ is a user-defined sensitive factor to leverage the effect of the STAC on the coupling weight. The smaller the α_0, the less sensitive the coupling weight is to the STAC. For example, $\alpha_0 = 0$ means that the STAC does not affect the coupling weight. By default, $\alpha_0 = 1$. The result of Equation 4.21 is denoted by $w_e^{(i,j)}$.

Optimal Coupling Weights

Here is the definition of the optimal coupling weight.

Definition 22. *The optimal coupling weight between two vehicles represents the difficulty encountered by the lower-priority vehicle to avoid a collision with the higher-priority vehicle when planning trajectory.*

The calculating of the optimal coupling weight between two vehicles is nontrivial. One way is to use reachability analysis to count how many switchable motion primitives of the lower-priority vehicle are valid. A motion primitive is valid if, after the vehicle has used this motion primitive, a trajectory of $H_p - 1$ time steps avoiding the reachable set of the higher-priority vehicle still exists. Note that here a trajectory of only $H_p - 1$ (but not H_p) time steps is required because the switchable motion primitive is the trajectory segment of the first time step, and thus one only needs to check whether a feasible trajectory can be found for the remaining time steps. The percentage of the invalid motion primitives of the lower-priority vehicle is the optimal coupling weight between those two vehicles.

Proposition 3. *If the optimal coupling weight of two vehicles is one, and their coupling is avoided by consideration of the reachable set of the higher-priority vehicle, the lower-priority vehicle will certainly encounter an infeasibility.*

Proof. Consider two vehicles i and j, with vehicle i having a higher priority. The optimal coupling weight being one means all switchable motion primitives of vehicle j are invalid, i.e., whichever motion primitive is executed, vehicle j will not find a feasible trajectory to avoid the reachable set of vehicle i. Therefore, vehicle j will certainly encounter an infeasibility. □

Based on Proposition 3, if the optimal coupling weight of two vehicles is one, one can force them to plan trajectories in sequence, avoiding considering the reachable set. This provides a good criterion to find which vehicles should plan trajectories one after another. However, the computation of the optimal coupling weights is time-consuming, as the trajectory planner is already used in this stage for pre-checking the validity of each switchable motion primitive. Nevertheless, the optimal coupling weights can be used as a benchmark to evaluate the proposed STAC-based coupling weights, as shown in Section 5.2.1.

4.2.4. Edge-Weighted Directed Coupling Graph Determination Algorithm

This subsection proposes an algorithm (Algorithm 4) to determine the edge-weighted directed couplings graph between all vehicles in NCSs and their coupling weights.

Algorithm 4 returns the coupling matrix $A_c \in \mathbb{R}^{N\times N}$ and the weighting matrix $A_w \in \mathbb{R}^{N\times N}$ of the edge-weighted directed coupling graph modeling the coupling situation of vehicles in NCSs based on the states $\boldsymbol{x}^{(i)}$, maximum acceleration $a_+^{(i)}$ and deceleration $a_-^{(i)}$, and lanelet l_i of each vehicle $i = 1, \ldots, N$. In addition, the adjacent lanelet relationship matrix A_l^R is needed to determine the lanelet relationship of the lanelets of two vehicles, and the adjacent lanelet critical point matrix A_l^P can be used to query the lanelet critical point of each pair of adjacent lanelets.

Line 1 initializes the coupling matrix A_c and the weighting matrix A_w. Lines 2 and 3 iterate all pairs of vehicles without repetition. Next, whether the selected two vehicles i and j are coupled is checked using the function $isCoupled$ defined in Equation 4.13. They are uncoupled if $isCoupled(i, j) = false$. In that case, the corresponding entries of the coupling matrix and weighting matrix will maintain their initial values, i.e., $A_c(i, j) = 0$ and $A_w(i, j) = 0$, and subsequent iteration starts. If they are coupled, a variable $higherP$ is initialized with $true$ (see line 5), assuming vehicle i has a higher priority over vehicle j. If that is not the case, its value will be changed to $false$ later.

If vehicles i and j have a rear-end collision possibility (line 6), their distances to the lanelet critical point will be calculated (line 7). The vehicle with a smaller distance to their lanelet critical point is the front vehicle and will have a higher priority. According to which vehicle is the front vehicle, their TTC t_{TTC} is computed using Equation 4.17 (see line 9 and line 11). After that, the STAC equals the TTC according to Equation 4.19, and the waiting time is zero. Recall that the usage of the waiting time is only necessary if two vehicles have a side-impact collision possibility.

If vehicles i and j have a side-impact collision possibility, whether they are moving side by side will be checked (line 15). According to Equation 4.15, two vehicles with a side-impact collision possibility move side by side only if their reference lanelets are the same lanelet, left-, right-, or merging-adjacent lanelets. If this is the case, the ACP is approximated as the middle point of their centers of gravity (line 16); otherwise, it will be the lanelet critical point (line 20). However, the lanelet critical point is still needed to check which vehicle has a higher priority if they move side by side: The vehicle closer to their lanelet critical point is the front vehicle and has a higher priority (see line 17 and line 18). After that, distances from both vehicles to the ACP are computed (line 22), and their TTA $t_{TTA}^{(i,j)}$ and $t_{TTA}^{(j,i)}$ are calculated using Equation 4.18 (line 23 and line 24). If vehicles do not move side by side, a higher priority is given to the vehicle with a smaller TTA (line 25). In line 26, the STAC

Algorithm 4: Edge-weighted directed coupling graph determination algorithm

Input: current time step k, $\forall i = 1, \dots, N : \boldsymbol{x}^{(i)}, a_+^{(i)}, a_-^{(i)}, l_i; A_l^R; A_l^P$
Output: coupling and weighting matrix: A_c, A_w

$A_c \leftarrow \mathbf{0}_{N\times N}$; $A_w \leftarrow \mathbf{0}_{N\times N}$; // assign initial value
for $i = 1, \dots, N-1$ **do**
  for $j = i+1, \dots, N$ **do**
    if $isCoupled(i,j,k)$ **then** // see Eq. 4.13
      $higherP \leftarrow true$; // assume vehicle i has a higher priority
      if $A_{CT}(i,j) == rearEnd$ **then** // see Eq. 4.15
        $d_{ist}^{LCP(i)}, d_{ist}^{LCP(j)} \leftarrow$ distances to lanelet critical point $A_l^P(l_i, l_j)$;
        if $d_{ist}^{LCP(i)} \leq d_{ist}^{LCP(j)}$ **then**
          $t_{TTC} \leftarrow t_{TTC}(v_0^{(i)}, a_-^{(i)}, v_0^{(j)}, a_+^{(j)}, d_{ist}^{(i,j)})$; // see Eq. 4.17
        else
          $t_{TTC} \leftarrow t_{TTC}(v_0^{(j)}, a_-^{(j)}, v_0^{(i)}, a_+^{(i)}, d_{ist}^{(i,j)})$; $higherP \leftarrow false$;
        end
        $\tau^{(i,j)} \leftarrow t_{TTC}$; $\tau_w^{(i,j)} \leftarrow 0$;
      else
        if $isSbS(i,j) == true$ **then** // check if move side by side
          $P_{ACP}^{(i,j)} \leftarrow$ get the middle point of vehicles i and j;
          $d_{ist}^{LCP(i)}, d_{ist}^{LCP(j)} \leftarrow$ get distances to lanelet critical point $A_l^P(l_i, l_j)$;
          if $d_{ist}^{LCP(i)} \geq d_{ist}^{LCP(j)}$ **then** $higherP \leftarrow false$;
        else
          $P_{ACP}^{(i,j)} \leftarrow A_l^P(l_i, l_j)$;
        end
        $d_{ist}^{ACP(i)}, d_{ist}^{ACP(j)} \leftarrow$ get distances to $P_{ACP}^{(i,j)}$;
        $t_{TTA}^{(i,j)} \leftarrow t_{TTA}(v_0^{(i)}, a_+^{(i)}, d_{ist}^{ACP(i)})$; // see Eq. 4.18
        $t_{TTA}^{(j,i)} \leftarrow t_{TTA}(v_0^{(j)}, a_+^{(j)}, d_{ist}^{ACP(j)})$;
        if $isSbS(i,j) == false \wedge t_{TTA}^{(i,j)} \geq t_{TTA}^{(j,i)}$ **then** $higherP \leftarrow false$;
        $\tau^{(i,j)} \leftarrow \max(t_{TTA}^{(i)}, t_{TTA}^{(j)})$; $\tau_w^{(i,j)} \leftarrow |t_{TTA}^{(i,j)} - t_{TTA}^{(j,i)}|$;
      end
      if $higherP == true$ **then** // vehicle i has a higher priority
        $A_c(i,j) \leftarrow 1$; $A_w(i,j) \leftarrow w_e(\tau^{(i,j)} + \tau_w^{(i,j)})$; // see Eq. 4.21
      else
        $A_c(j,i) \leftarrow 1$; $A_w(j,i) \leftarrow w_e(\tau^{(i,j)} + \tau_w^{(i,j)})$;
      end
    end
  end
end
return A_c, A_w

$\tau^{(i,j)}$ is computed by taking the maximum one of their TTA (see Equation 4.19), and the waiting time $\tau_w^{(i,j)}$ is calculated according to Equation 4.20.

Finally, according to which vehicle has a higher priority, the corresponding entry of the coupling matrix A_c will be changed to 1, and the same entry of the weighting matrix A_w will be assigned a weight calculated by Equation 4.21 (see line 29 and line 31).

4.2.5. Breaking of Coupling cycles

A directed coupling graph can be easily built based on the coupling matrix returned by Algorithm 4. However, the obtained directed graph is not guaranteed to be acyclic. In priority-based trajectory planning, the coupling graph must be acyclic. In sequential trajectory planning, vehicles in a coupling cycle will mutually wait for others, so no one starts to plan; in parallel trajectory planning, vehicles in a coupling cycle will mutually avoid collisions with others, such as by considering the reachable sets of others, increasing the possibility of deadlocks. In complex traffic scenarios, such as at intersections, coupling cycles may appear more often. For example, if Algorithm 4 is applied to the traffic scenario in Fig. 3.1, a coupling matrix would be obtained, whose corresponding directed coupling graph is shown in Fig. 4.10. As one may notice, there is a coupling cycle in the graph (the red cycle in the figure). Therefore, when planning trajectories, vehicle 3 will wait for vehicle 2, vehicle 4 will wait for vehicle 3, and vehicle 2 will wait for vehicle 4, causing no one really starts to plan its trajectory.

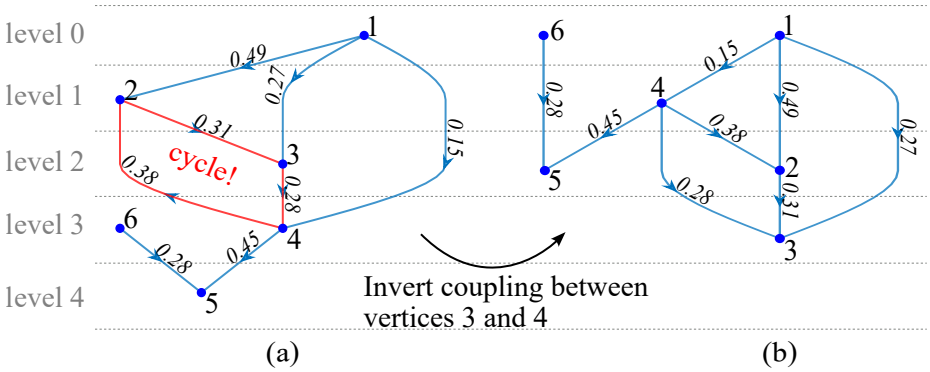


Figure 4.10.: Break a coupling cycle in a directed coupling graph. (a) Coupling graph with a cycle. (b) Obtained DAG by inverting the coupling direction between vertices 3 and 4.

To avoid the above-mentioned problem, this section proposes a simple strategy to break cycles in directed graphs while minimizing its impact on collision avoidance. The couplings corresponding to the broken edges must still be considered by vehicles, i.e., cycles cannot be simply broken by removing some coupling edges, as in this way, the removed couplings would possibly cause collisions. Firstly, a cycle detection algorithm for directed graphs is needed. Many such algorithms exist. Interested readers are referred to [Tar72, Sha81], which are the most commonly used and efficient algorithms to find cycles in directed graphs. Both algorithms find cycles by finding strongly connected components in directed graphs, as cyclic directed graphs must contain strongly connected subgraphs. After cycles are found, they will be broken iteratively by inverting the directions of the coupling edges. During each iteration, the coupling edge with the smallest coupling weight is inverted since the corresponding two vehicles have the lowest possibility of colliding. After inverting each coupling direction, cycles are detected again. This procedure is repeated until no cycle exists. The results of the proposed strategy are shown in Fig. 4.10(b). The only cycle $c(2,3,4)$ is broken by inverting the direction of the edge connecting vertices 3 and 4 as it has the lowest weight, i.e., $w_e^{(3,4)} < w_e^{(2,3)} < w_e^{(2,4)}$.

Although the proposed strategy works well in breaking coupling cycles, one may want to avoid them at the source. This is possible with marginal adaptions of Algorithm. 4. As mentioned before, the occurrence of coupling cycles is more possible at road intersections. Strictly speaking, the following proposition exists.

Proposition 4 (Coupling cycles only at road intersections)**.** *If Algorithm 4 is used to determine coupled vehicles, coupling cycles could only appear at intersections. No coupling cycle could appear at all other common road types, such as left-adjacent, right-adjacent, forking-adjacent, and merging-adjacent lanelets.*

Proof. This can be proved by contradiction. According to lanelet types, the following two cases exist.

- For vehicles at the same, left-, right-, longitudinal-, or forking-adjacent lanelets, the front vehicles will always have higher priorities. Here $f_{\{i,j\}}$ is used to denote that vehicle i is in front of vehicle j. If there is a coupling cycle between vehicles $\{v_1, v_2, \ldots, v_m\}$, the following conclusions hold true: $f_{\{1,2\}}, f_{\{2,3\}}, \ldots, f_{\{m-1,m\}}$, and $f_{\{m,1\}}$. It is clear that the last one $f_{\{m,1\}}$, indicating that vehicle m is in front of vehicle 1, contradicts others. However, one exception is the circular road (or also called ring road). In case of a such road, coupling cycle may still occur, and thus the proposed cycle braking strategy can be used.
- For vehicles at two merging lanelets, they have the same ACP, i.e., the merging point. If there is a coupling cycle between vehicles $\{v_1, v_2, \ldots, v_m\}$, the following conclusions hold true: $t_{TTA}^{(1)} < t_{TTA}^{(2)} < \cdots < t_{TTA}^{(m-1)} < t_{TTA}^{(m)}$, and $t_{TTA}^{(m)} < t_{TTA}^{(1)}$. The last inequation obviously contradicts others.

□

For a mixture of lanelet types, the proof may not be strictly true. For example, in Fig. 4.8, lanelets $\{l_1, l_2, l_3, l_4\}$ are a mixture of merging-, left-, and right-adjacent lanelets. Normally, for vehicles at lanelets 1 and 2, vehicles that are closer to point A will have higher priorities. For vehicles at lanelets 3 and 4, the lanelet critical point is point C. For merging-adjacent lanelets 2 and 3, the lanelet critical point is point B. The proof is no longer true due to the different lanelet critical points. In this case, the trueness of the proof is kept if only one lanelet critical point is used for vehicles at those lanelets, e.g., point B. This compromise is acceptable, as in most cases, the reference points of such a mixture of different lanelet types are close to each other.

One may want to avoid coupling cycles essentially. According to Proposition 4, coupling cycles will never appear if they can be avoided at intersections. One straightforward way is to assign lower priorities to vehicles entering intersections later, i.e., vehicles newly to intersections are always assigned lower priorities.

Proposition 5 (Guarantee of no coupling cycle at intersections)**.** *Suppose vehicles at an intersection have no coupling cycle at the initial time step. In that case, coupling cycles will never appear if vehicles newly entering the intersection are assigned lower priorities than all vehicles are are already at the intersection.*

Proof. At the initial time step, the coupling graph of the vehicles at the intersection is a DAG since there is no coupling cycle. The assignment of a lower priority to each vehicle newly to the intersection equals adding a leaf to the existing DAG. As leaves have no out-edge, adding leaves will never build strongly connected components, i.e., adding leaves will never build coupling cycles to a DAG. □

Based on the soundness of Proposition 5, coupling cycles can be avoided if there is no coupling cycle at the initial time step. If this is not the case, the proposed strategy to break coupling cycles will be used at and only at the initial time step.

4.2.6. Priority Assignment Strategy

Once there is no coupling cycle in the DAG, one can straightforwardly assign priorities to vehicles. As introduced in Section 2.1, each vertex has a level in the DAG. Recall that the level of a root is zero. Priorities are assigned to vehicles according to the levels of the corresponding vertices in the DAG:

$$p^{(i)} = l^{(i)} + 1,$$

where $p^{(i)}$ is the priority of vehicle i, and $l^{(i)}$ is the level of vertex i. Note that priorities are positive integers $p^{(i)} \in \mathbb{N} = \{1, 2, \dots\}$, with a smaller value indicating

a higher priority. In this way, the value of a vehicle's priority equals the value of its computation level.

It is worth remarking that this priority assignment strategy can also be used after grouping vehicles in the next section. Each vehicle's priority equals its computation level in its group. The strategy proposed in Section 4.4.3 decouples vehicles, leading to a reduced number of couplings. However, priorities can still be assigned according to the new computation levels of vehicles after decoupling.

4.3. Formulation of Different Groups

Vehicles are split into groups, with vehicles in the same group planning trajectories sequentially while in different groups planning in parallel. In this section, the vehicle grouping problem is converted into a graph partitioning problem – or more specifically, into a min-cut clustering problem [JMN93] – based on the coupling matrix A_w obtained from the previous section. To further reduce the objective value of the graph partitioning algorithm, a simple but effective graph merging algorithm is proposed. Note that the terms cluster(s) and subgraph(s) will be used interchangeably.

According to Definition 13, the number of computation levels n_{CL}^i of a group i has the following relationship with the depth of the corresponding DAG modeling its group: $n_{CL}^i = d(G_i) + 1$, where $i \in \{1, \ldots, k\}$ is the group or subgraph index. The larger the number of computation levels of a group, the more vehicles will be waited by the vehicles on the last computation level. Assume all vehicles need the same time to plan a trajectory. A group's total trajectory planning time will grow linearly with the number of its computation levels. Recall Problem 1 in Section 2.1.2. Here, another constraint 4.22d will be added to limit the number of computation levels of each group. For the sake of completeness, the whole (adapted) min-cut clustering problem is rewritten in Problem 3.

Problem 3 (Adapted min-cut clustering problem)**.** *Given an edge-weighted DAG $G = (\mathcal{V}, \mathcal{E})$ with edge weight w_e for each edge $e \in \mathcal{E}$, find a partition $\Gamma = \{\mathcal{V}_1, \ldots, \mathcal{V}_k\}$ of $\mathcal{V}$ that solves 4.22a while satisfying constraints from 4.22b to 4.22d.*

$$\Gamma = \arg\min_{\Gamma} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \sum_{e \in \mathcal{E}_{i,j}} w_e \tag{4.22a}$$

subject to

$$\mathcal{V}_1 \cup \mathcal{V}_2 \cup \cdots \cup \mathcal{V}_k = \mathcal{V}, \tag{4.22b}$$

$$\mathcal{V}_i \cap \mathcal{V}_j = \emptyset, \forall i, j \in \{1, \ldots, k\}, i \neq j, \tag{4.22c}$$

$$d(G_i) \leq d_{max}, \forall i \in \{1, \ldots, k\}. \tag{4.22d}$$

The last constraint 4.22d ensures that the depth of each subgraph is smaller than or equal to a defined integer $d_{max} \in \mathbb{N}$ and thus effectively limits the total trajectory planning time of each group.

4.3.1. Graph Partitioning Algorithm

To the best of the author's knowledge, no suitable algorithm has been proposed to solve Problem 3. For this purpose, Stoer's algorithm (Algorithm 1) detailed in Section 3.3.1 will be adapted in this subsection.

Adaption of Stoer's Algorithm

As one can see, Stoer's algorithm does not consider Constraint 4.22d in Problem 3. In addition, it cuts a graph into two instead of multiple parts. Stoer's algorithm will be used iteratively to overcome those two drawbacks. The adapted Stoer's algorithm is given in Algorithm 5.

Algorithm 5 calls the *MinimumCut* procedure, which is the main body of Stoer's algorithm, iteratively. Initially, the given graph is set as the deepest subgraph G_d (line 1), i.e., the subgraph with the largest depth. In line 2, a variable G_{sub} is created to store the resulting subgraphs. During each iteration, the deepest subgraph G_d will be cut by Stoer's algorithm (line 6), and the two resulting subgraphs will be stored (line 8). The final subgraphs will be returned until no subgraph has a depth larger than the defined value d_{max}.

Proposition 6. *For a DAG with N vertices, Algorithm 5 is guaranteed to terminate after at most $N - d_{max} - 1$ iterations.*

Proof. Consider the worst-case scenario. In the worst case, Algorithm 1 cuts only one vertex each time. The maximum depth of a DAG with N vertices is $N - 1$.

Algorithm 5: Adapted Stoer's algorithm

Input: target DAG G, maximum depth of each subgraph: d_{max}
Output: resulting subgraphs: $G_1, \ldots, G_k$

1 $G_d \leftarrow G$; // store the deepest subgraph
2 $G_{sub} \leftarrow \{G_d\}$; // store subgraphs
3 **while** $\underline{d(G_d) > d_{max}}$ **do**
4 $A_{w,d} \leftarrow A_w(G_d)$; // weighting matrix of the deepest subgraph
5 $\mathcal{V}_d \leftarrow \mathcal{V}(G_d)$; // vertices of the deepest subgraph
6 $\mathcal{V}_1, \mathcal{V}_2 \leftarrow MinimumCut(\mathcal{V}_d, A_{w,d})$; // see Alg. 1
7 $G_{sub} \leftarrow G_{sub}/G_d$; // delete the original graph G_d
8 $G_{sub} \leftarrow G_{sub} \cup \{G(\mathcal{V}_1), G(\mathcal{V}_2)\}$; // add new subgraphs
9 $G_d \leftarrow$ find the deepest subgraph in G_{sub};
10 **end**
11 **return** $\underline{G_1, \ldots, G_k}$

Therefore, the remaining graph has at most $d_{max}+1$ vertices after $N-d_{max}-1$ times of cutting. Since the depth of the remaining graph is at most d_{max}, the condition in line 3 is unsatisfied, leading to the termination of the algorithm. The results are a graph with $d_{max}+1$ vertices and $N-d_{max}-1$ trivial graphs (a graph is trivial if it has only one vertex). □

Fig. 4.11 supports the proof of the soundness of Proposition 6. It illustrates the cutting results of a DAG with 5 vertices where d_{max} is set to 2. Algorithm 5 terminates after two cuts: Vertex 5 is cut first while vertex 4 is cut second. The cost of the first cut is 0.1, and the cost of the second cut is 0.2. Three subgraphs are obtained: $\{\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3\} = \{\{v_5\}, \{v_4\}, \{v_1, v_2, v_3\}\}$ with a total of $cost_{total} = w_e^{(4,5)} + w_e^{(3,4)} = 0.1 + 0.2 = 0.3$.

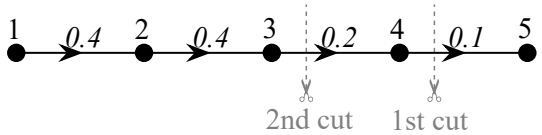


Figure 4.11.: Cut a simple edge-weighted DAG with 5 vertices. Algorithm 5 terminates after two cuts.

4.3.2. Graph Merging Algorithm

As one may notice, the subgraphs $\mathcal{V}_1$ and $\mathcal{V}_2$ in the example mentioned above (see Fig. 4.11) can be merged into one graph without violating the maximum depth

constraint, as it results in a two-vertex graph with a depth of $2(\leq d_{max})$. This way, the total cost will be reduced from 0.3 to 0.2, as the cut edge $e_{4,5}$ is restored. Inspired by this, a graph merging algorithm is proposed to reduce further the total cost of the adapted Stoer's algorithm.

Algorithm 6: Graph merging algorithm

Input: subgraphs $G_1, \ldots, G_{k_{old}}$, maximum allowed depth of subgraphs d_{max}
Output: new subgraphs: $G_1, \ldots, G_k$

1 $g_{merged} \leftarrow \emptyset$; `// store the indices of merged subgraphs`
2 **for** $i = \{1, \ldots, k_{old} - 1\}/g_{merged}$ **do**
3 $b_{max} \leftarrow 0$; `// record the benefit of merging two subgraphs`
4 **for** $j = \{i+1, \ldots, k_{old}\}/g_{merged}$ **do**
5 $G_{tmp} \leftarrow$ merge G_j into G_i; `// a temporary graph`
6 **if** $d(G_{tmp}) \leq d_{max}$ **then**
7 $b_{tmp} \leftarrow$ benefit of merging G_i and G_j;
8 **if** $b_{tmp} > b_{max}$ **then**
9 $b_{max} \leftarrow b_{tmp}$;
10 $j_{best} \leftarrow j$;
11 **end**
12 **end**
13 **end**
14 **if** $b_{max} > 0$ **then** `// check if merging is really beneficial`
15 $G_i \leftarrow$ merge $G_{j_{best}}$ into G_i;
16 $g_{merged} \leftarrow g_{merged} \cup j_{best}$; `// add` j_{best}
17 **end**
18 **end**
19 $G_1, \ldots, G_k \leftarrow$ rearrange the indices of subgraphs;
20 **return** $G_1, \ldots, G_k$

Algorithm 6 takes the resulting subgraphs $G_1, \ldots, G_{k_{old}}$ of Algorithm 5 and the maximum allowed depth d_{max} as input. It iterates over all pairs of subgraphs using two for-loops (line 2 and line 4). The first for-loop picks a target subgraph G_i (line 2). Next, the benefits of merging G_i with each subgraph $G_j, j = i+1, \ldots, k_{old}$ will be computed separately to find the best merging candidate for G_i (lines 4 to 13). Here, the benefit refers to the reduced cost of each merging. The one with the maximum benefit will be chosen, and the corresponding subgraph $G_{j_{best}}$ will be merged into the target subgraph G_i (line 15). The index j_{best} of the merged subgraph $G_{j_{best}}$ will be added to the list g_{merged} to ensure that it will not be iterated again (line 16). If, however, no suitable merging candidate for G_i exists, the variable b_{max} recording the maximum benefit will keep its initial value, zero. In this case, the if-condition in

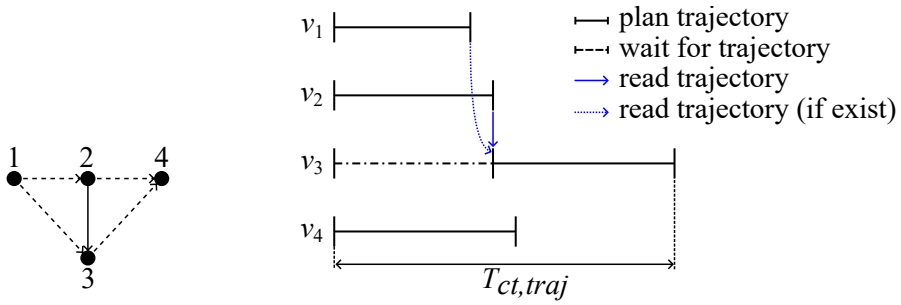
line 14 is not satisfied; thus, no subgraph will be merged into G_i, and the algorithm will iterate to the next target subgraph. This procedure is repeated until all pairs of subgraphs are checked. At the end of the algorithm, the indices of the subgraphs will be rearranged (line 19) and returned (line 20).

4.3.3. Example with Four Vehicles

Let us revisit scenario 2 in Fig. 4.7(b). If Algorithms 5 and 6 are applied to it with the maximum depth of subgraphs setting to 2, one can get the results shown in Fig. 4.12(a). The resulting subgraphs are: $G_1 = \{1\}, G_2 = \{2, 3\}, G_3 = \{4\}$, meaning vehicles inside group 2 plan trajectories in parallel to other two groups of vehicles. Their trajectory planning order is depicted in (b). Vehicles 1, 2, and 4 plan trajectories simultaneously, while vehicle 3 must wait for vehicle 2. It is assumed that the more couplings a vehicle considers, i.e., the higher in-degree a vertex has, the longer its trajectory planning time will be, which is reasonable since it will be harder to find a feasible trajectory in this case. As the in-degree of vertex 1 is zero, vehicle 1 finishes its planning first. After some time, vehicle 2 with one in-degree has also finished; then, it communicates its predicted trajectory to vehicle 3. Vehicle 3, however, should additionally consider its cross-group coupling with vehicle 1. As vehicle 1 has already finished its planning, vehicle 3 can read its predicted trajectory. If that is not the case, i.e., if vehicle 1 takes a longer time to plan a trajectory than vehicle 2, vehicle 3 must consider vehicle 1's reachable set. This is why the corresponding blue line in (b) is depicted as dashed but not solid. As one can see, in this example, the total trajectory planning time $T_{ct,traj}$ depends on the trajectory planning time of the second group, i.e., depends on the group with the largest number of computation levels. Note that this is not always the case, as the trajectory planning time of each vehicle depends highly on the trajectory planner used and the constraints considered.

4.4. Feasibility and Deadlocks

This section deals with feasibilities and deadlocks. Section 4.4.1 presents a strategy to improve feasibilities by considering the emergency braking maneuvers of lower-priority vehicles. Section 4.4.2 proposes a fallback strategy, which will be used when a vehicle cannot find a feasibility trajectory. Besides infeasibilities, one must prevent deadlocks. Section 4.4.3 introduces an effective strategy to prevent deadlocks, especially at intersections where deadlocks occur easily.



(a) Directed coupling graph. (b) Trajectory planning order.

Figure 4.12.: The cut edges are depicted in dashed lines while the uncut edge is shown in solid line. $T_{ct,traj}$ is the total trajectory planning time of the four vehicles.

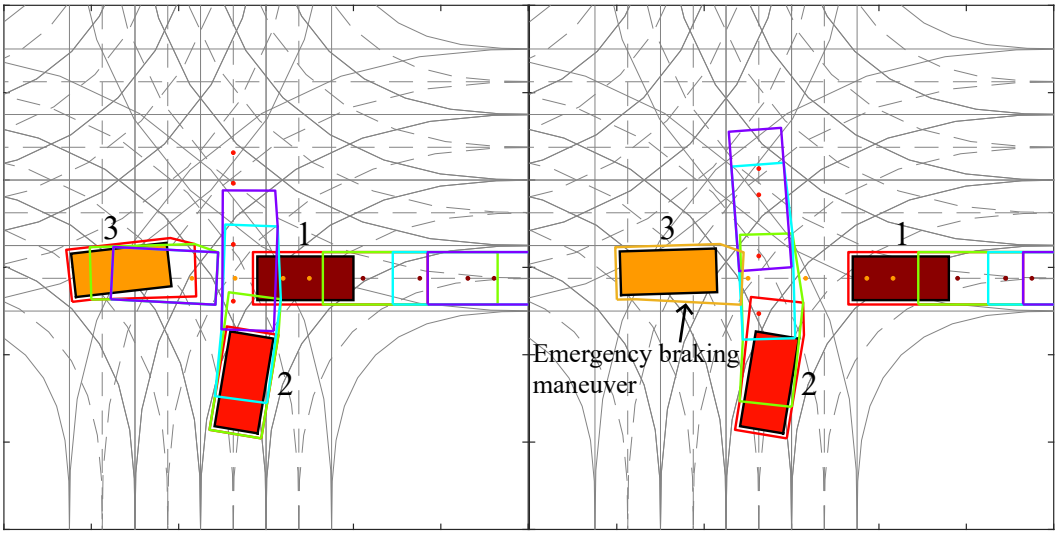
4.4.1. Improvement of Feasibility

In trajectory planning, a feasible solution is a solution that satisfies all constraints of Problem 2. Although the proposed priority assignment approach overcomes the drawbacks of some state-of-the-art approaches, infeasibilities could still occur due to the incompleteness problem of the priority-based trajectory planning. This section describes a strategy to improve feasibilities.

The author in [Su22] suggested letting vehicles with higher priorities consider the currently occupied sets of their coupled vehicles with lower priorities as static obstacles when trajectory planning. Henceforth, the currently occupied set of vehicle i will be denoted as $\mathcal{O}_c^{(i)}$. This strategy improves the feasibilities of lower-priority vehicles to some degree. However, there are some infeasibilities that cannot be avoided by this strategy. In Fig. 4.13, three vehicles are at an intersection. Vehicle 1 has the highest priority and moves rightwards; vehicle 3 has the lowest priority and also moves rightwards; vehicle 2 moves upwards. At time step k (Fig. 4.13(a)), all of them find feasible trajectories. At the first prediction horizon, vehicle 2 stops to avoid the predicted occupied set of vehicle 1 (see their red polygons). At the second prediction horizon, vehicle 2 is able to move upwards as vehicle 1 is not in its way anymore (see their green polygons). Vehicle 3, therefore, brakes until it stops to avoid a collision with vehicle 2 (see their green polygons). At the third and fourth prediction horizons, vehicle 3 keeps at a standstill. Note that the azure polygon of vehicle 3 is not visible as the purple polygon covers it. At the next time step (Fig. 4.13(b)), vehicle 2 turns left more sharply to follow its reference trajectory (shown in dots) more tightly. As a result, although vehicle 3 takes an emergency braking maneuver (see the yellow polygon), it can only avoid a collision at the first prediction horizon. Collisions will occur in the remaining prediction horizons. One may ask why vehicle

2 turns left more sharply instead of using its previous plans. The reason is that the used trajectory planning algorithm, i.e., the A* search algorithm (see Section 2.4.3), estimates the cost-to-go approximately (i.e., its lower bound). At the next time step, the approximated cost-to-go can be decreased using a more sharply left-turn maneuver. Therefore, even if a vehicle has the highest priority and thus does not need to consider others' plans, it is still possible that this vehicle would change its plan at the next time step.

To avoid the above-mentioned type of collisions, this thesis suggests letting higher-priority vehicles consider the occupied sets of the emergency braking maneuvers of coupled vehicles with lower priorities. In this case, vehicle 2 will stop since it needs to avoid the occupied set of the emergency braking maneuver of vehicle 3, and vehicle 3 will thus move rightwards and cross the intersection before vehicle 2. The emergency braking trajectory of vehicle i at time step k is denoted by $\pi^{(i)}_{eb,([k,k+H_p]|k)}$, and its occupied set is thus $\mathcal{O}(\pi^{(i)}_{eb,([k,k+H_p]|k)})$.



(a) Time step k. (b) Time step k+1.

Figure 4.13.: Prediction horizon $H_p = 4$. Red, green, azure, and purple polygons are the predicted occupied sets of the first, second, third, and last prediction horizon, respectively. Reference trajectories are shown in dots.

Proposition 7 (Guarantee of Feasibility)**.** *For each coupled pair of vehicles plan trajectories in sequence, if the higher-priority one can find a trajectory avoiding*

the occupied set of the emergency braking maneuver of the lower-priority one, it is guaranteed that the lower-priority vehicle can certainly find a trajectory to avoid collision with the higher-priority vehicle.

Proof. Consider arbitrary two coupled vehicles i and j planning trajectories in sequence. Assume that vehicle i has a higher priority without loss of generality. At an arbitrary time step k, vehicle i calculates the emergency braking trajectory $\pi^{(j)}_{eb,([k,k+H_p]|k)}$ of vehicle j and then plans a trajectory $\pi^{(i)}_{([k,k+H_p]|k)}$ avoiding the occupied set $\mathcal{O}(\pi^{(j)}_{eb,([k,k+H_p]|k)})$ of this emergency braking trajectory:

$$\mathcal{O}(\pi^{(i)}_{([k,k+H_p]|k)}) \cap \mathcal{O}(\pi^{(j)}_{eb,([k,k+H_p]|k)}) = \emptyset.$$

Next, vehicle j is expected to avoid collisions with vehicle i. Obviously, vehicle j can at least plan an emergency braking trajectory $\pi^{(j)}_{eb,([k,k+H_p]|k)}$ to avoid collisions with vehicle i since the above equation holds. □

From the soundness of Proposition 7, it follows that the proposed strategy improves the feasibilities of lower-priority vehicles. One downside is that it sacrifices the trajectory quality of higher-priority vehicles since they need to consider additional constraints. Note that this feasibility improving strategy is only applied to coupled vehicles with side-impact collision possibilities. Suppose two vehicles have a rear-end collision possibility. In that case, considering the rear vehicle's emergency braking maneuver may cause the front vehicle's infeasibility if they are very close to each other. This is the case when the occupied set of the emergency braking maneuver of the rear vehicle intersects with the currently occupied set of the front vehicle. Therefore, if two vehicles have a rear-end collision possibility, the front vehicle will consider the currently occupied set of the rear vehicle, as suggested in [Su22].

4.4.2. Dealing with Infeasibilities

Although the proposed strategy largely improves feasibilities, infeasibilities may still occur. In this section, the nature of MPC – plan for the future – is exploited to deal with infeasibilities.

Under the MPC framework, at each time step, vehicles plan trajectories for H_p time steps, and only the trajectory segment in the current time step will be used. If, for example, vehicle i cannot find a feasible trajectory at time step k, it can use its previously planned trajectory $\pi^{(i)}_{([k-1,k+H_p-1]|(k-1))}$. This concept has been mentioned in [GA19] but without any further explanation. Each planned trajectory consists of H_p components. Each component corresponds to one time step. The first component $\pi^{(i)}_{([k-1,k]|(k-1))}$ will be discarded as it has already been used, and the last component

$\pi^{(i)}_{([k+H_p-2,k+H_p-1]|(k-1))}$ will be repeated, making the number of components equals to H_p. The new trajectory will be

$$\pi^{(i)}_{([k,k+H_p]|k)} = \pi^{(i)}_{([k,k+H_p-1]|(k-1))} \cup \pi^{(i)}_{([k+H_p-2,k+H_p-1]|(k-1))}. \tag{4.23}$$

This one-step-shifted previously planned trajectory will be called a fallback trajectory, denoted as $\pi^{(i)}_{fb,([k,k+H_p]|k)}$ for vehicle i, or $\pi^{(i)}_{fb}$ in short. To guarantee that the fallback trajectory of a vehicle is collision-free with other vehicles, the author in [Su22] suggested letting all other vehicles discard their currently planned trajectories and use their fallback trajectories. However, this fallback strategy, henceforth called the global fallback strategy, may degrade the traffic performance as all vehicles must use their previously planned trajectories instead of planning new trajectories based on the current traffic situation.

To mitigate the drawback of the global fallback strategy, this thesis proposes a local fallback strategy, i.e., only let weakly coupled vehicles (see Definition 8) with the vehicle that triggers a fallback use their fallback trajectories. That is, if vehicle i triggers a fallback, vehicle j must also use its fallback trajectory, $\forall j \in \mathcal{V}^{(i)}_{wc}$. For example, in Fig. 4.10(b), if vehicle 2 triggers a fallback, all other vehicles should also use their fallback trajectories since all vertices are in a weakly connected component. In Fig. 4.14(b), if vehicle 1 triggers a fallback, only vehicle 1 itself and vehicle 3 need to use their fallback trajectories. This way, the other four vehicles can plan their new trajectories considering the new traffic situation.

Proposition 8. *The local fallback strategy guarantees the freeness of collisions in NCSs for an infinite time if all vehicles can find feasible trajectories at the first time step.*

Proof. First, $CFT_{(k)} \in \mathbb{B}$ will be used to denote whether all vehicles in the NCS have collision-free trajectories for H_p time steps at time step k, which can be either newly planned trajectories or fallback trajectories. The simplest example is that all vehicles are at a standstill at time step k, and their planned trajectories at this time step will keep them at standstill. In this case, $CFT_{(k)} = true$.

Mathematical induction is used to prove this proposition as follows:

Basis: As assumed in the proposition, all vehicles can find feasible trajectories at the first time step, meaning that all of them have collision-free trajectories for H_p time steps at time step 1. Therefore, $CFT_{(1)} = true$.

Inductive step: Assume $CFT_{(k)} = true$. Next, $CFT_{(k+1)}$ will be proved to be true. Two cases are classified:

1. If no vehicle triggers a fallback at time step $k+1$: All vehicles can find feasible trajectories since no vehicle needs to trigger a fallback, which are collision-free for H_p time steps. Therefore, $CFT_{(k+1)} = true$.
2. If at least one vehicle triggers a fallback: According to the local fallback strategy, vehicles that are weakly coupled with the vehicles triggering fallbacks must also use their fallback trajectories. Collisions between arbitrary two vehicles can be classified into three cases according to their trajectory types.
   a) Collisions between two vehicles that both use their fallback trajectories at time step $k+1$: As $CFT_{(k)} = true$, their trajectories at time step k are collision-free for H_p time steps. Since the fallback trajectories are generated by deleting the trajectory segment of the first time step and repeating the trajectory segment of the last time step, one only needs to check whether the last trajectory segments of both vehicles are collision-free, which is apparently the case as they are the same as the second last trajectory segments. Therefore, the resulting fallback trajectories are still collision-free for H_p time steps.
   b) Collisions between two vehicles that one uses its fallback trajectory while another one plans a new trajectory at time step $k+1$: They are not coupled and even not weakly coupled at time step $k+1$; otherwise, both of them must use their fallback trajectories. According to Definition 16 about *coupled*, there is no collision possibility between them in H_p time steps. Therefore, their trajectories must be collision-free for H_p time steps.
   c) Collision between two vehicles that both plan new trajectories at time step $k+1$: They plan new trajectories instead of using fallback trajectories, meaning that they are firstly not coupled or weakly coupled with vehicles triggering fallbacks, and secondly, they can find feasible trajectories (instead of triggering fallbacks), which are collision-free for H_p time steps.

   As the trajectories of arbitrary two vehicles are collision-free for H_p time steps at time step $k+1$, it concludes that $CFT_{(k+1)} = true$.

□

Semi-Fallback

Since the couplings are changed dynamically between in- and cross-group couplings, a vehicle may suddenly need to avoid the reachable set of another vehicle, which is sometimes impossible because they plan trajectories in sequence before and thus are very close to each other. Fallbacks triggered by vehicles at a standstill will be called semi-fallbacks. Semi-fallbacks could occur, for example, at road intersections when some vehicles (called old vehicles) are waiting to cross the intersections and another vehicle suddenly arrives at the intersection from behind with a very short distance

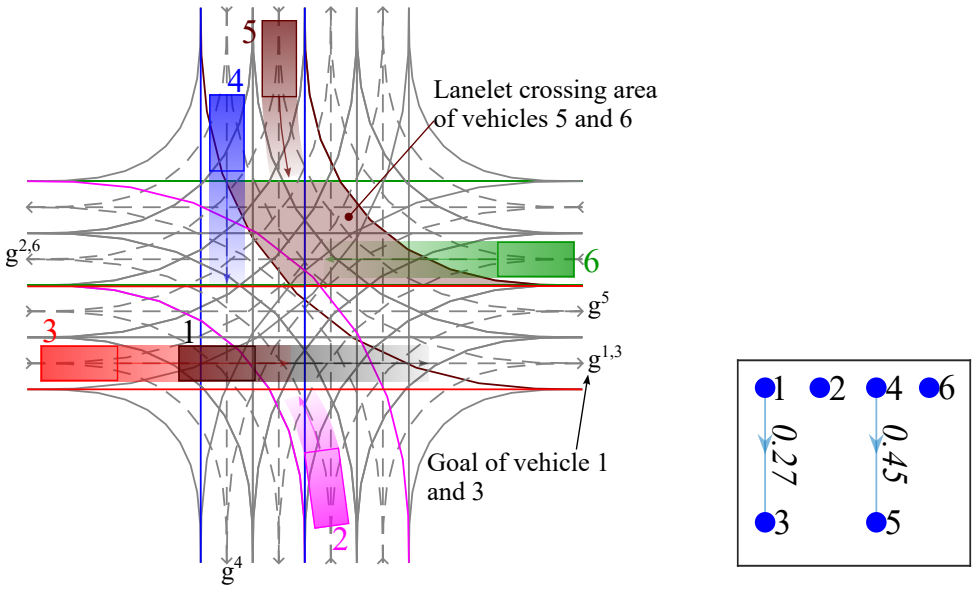
to one of the old vehicles; as a possible result, the vehicle grouping algorithm groups those two vehicles together, turning a previous in-group coupling between the old vehicles to a cross-group coupling. To avoid such fallbacks, one can forbid fallbacks triggered by vehicles at a standstill, and let them stay at a standstill, as they do not actively endanger others. To avoid collisions with vehicles at a standstill by vehicles with higher priorities, one can let higher-priority vehicles consider either the emergency braking maneuvers of their coupled vehicles with lower priorities (if they have side-impact collision possibilities) or the currently occupied sets of them (if they have rear-end collision possibilities).

4.4.3. Prevention of Deadlocks

Deadlocks occur when vehicles mutually block each other. As said in [PBL+19], the prevention of deadlocks is nontrivial and requires a compromise between computation time and traffic performance. This section proposes a simple but efficient and effective strategy for preventing deadlocks, which sacrifices acceptable traffic performance. It will be called the vehicle decoupling strategy, as it prevents deadlocks by decoupling vehicles. Coupled vehicles are decoupled in the sense that they can plan trajectories simultaneously without causing the prediction inconsistency problem.

There are many reasons why deadlocks occur. For example, as previously mentioned, deadlocks could happen if higher-priority vehicles consider the previous trajectories of lower-priority vehicles to increase feasibility (see vehicles 5 and 6 in Fig. 3.1). In addition, the occurrence of deadlocks highly relates to the priority assignment strategy. For some complex scenarios, it is hard to assign priorities to ensure both collision- and deadlock-free trajectories. Let us look at vehicles 1, 2, 3, and 4 in Fig. 3.1. If all the four vehicles plan trajectories in sequence, a deadlock could happen. According to Algorithm. 4, it is reasonable to assume that their priorities have the following relationship: $p_1 < p_2 < p_3 < p_4$, i.e., vehicle 1 has the highest priority while vehicle 4 has the lowest priority. At each time step, vehicle 1 plans its trajectory first. Next, vehicle 2 plans its trajectory while avoiding colliding with vehicle 1. As expected, in the beginning, vehicle 2 waits for a while until vehicle 1 has passed in front of it. Therefore, its predicted trajectory is much shorter. Then, vehicle 3 starts planning. In order to avoid a collision with vehicle 2, vehicle 3 stops shortly in front of the predicted trajectory of vehicle 2 at the last prediction step. Finally, vehicle 4 plans its trajectory and stops shortly in front of the predicted trajectories of both vehicles 2 and 3, blocking the way of vehicle 2. Not surprisingly, vehicles 2, 3, and 4 will stop in front of each other after several time steps.

To prevent this kind of deadlocks, this thesis suggests forbidding vehicles with lower priorities to enter their lanelet crossing areas before their coupled vehicles with higher priorities have left, or after they are no longer coupled. The lanelet crossing area of



(a) Lanelet crossing area. (b) Coupling graph.

Figure 4.14.: (a) Vehicles and their lanelet boundaries are depicted in the same color. (b) Coupling graph after using the vehicle decoupling strategy.

two vehicles is the intersecting area of their lanelet boundaries. A visualization is given in Fig. 4.14, where the lanelet crossing area of vehicles 5 and 6 are marked in brown. Vehicle 5, with a lower priority, is not allowed to enter this area. Therefore, its predicted trajectory ends shortly before this area. The occupied set of the lanelet crossing area of vehicles i and j is denoted by $\mathcal{O}_{lca}^{(i,j)}$.

Proposition 9 (Decoupling of vehicles)**.** *Collisions between two coupled vehicles at crossing- or merging-adjacent lanelets are impossible if one of them is not allowed to enter their lanelet crossing area. Therefore, they do not need to be coupled, i.e., they can be decoupled.*

Proof. Consider arbitrary two coupled vehicles i and j at crossing- or merging-adjacent lanelets at time step k. Assume vehicle j is not allowed to enter their lanelet crossing area without loss of generality. The occupied set of the area bounded by vehicle $\square$'s lanelet boundaries is denoted by $\mathcal{O}_{lb}^{(\square)}$. Analog, the occupied set of their lanelet crossing area is denoted by $\mathcal{O}_{lca}^{(i,j)}$. Next, it will be formally proved that if both vehicles are assumed not to cross their lanelet boundaries, collisions between them are impossible even if both vehicles do not consider the predicted trajectories of each other.

As both of the vehicles are assumed to always stay inside their lanelet boundaries, the follows hold true:

$$\mathcal{O}(\pi^{(i)}_{([k,k+H_p]|k)}) \subseteq \mathcal{O}^{(i)}_{lb} \text{ and} \tag{4.24a}$$

$$\mathcal{O}(\pi^{(j)}_{([k,k+H_p]|k)}) \subseteq \mathcal{O}^{(j)}_{lb}. \tag{4.24b}$$

$\mathcal{O}^{(\square)}_{\pi}$ will be used to simplify the denotation $\mathcal{O}(\pi^{(\square)}_{([k,k+H_p]|k)})$. As vehicle j is not allowed to enter the lanelet crossing area, it holds that

$$\mathcal{O}^{(j)}_{\pi} \cap \mathcal{O}^{(i,j)}_{lca} = \emptyset. \tag{4.25}$$

After replacing $\mathcal{O}^{(i,j)}_{lca}$ in Equation 4.25 with $\mathcal{O}^{(i)}_{lb} \cap \mathcal{O}^{(j)}_{lb}$, the following transformation can be established:

$$\mathcal{O}^{(j)}_{\pi} \cap (\mathcal{O}^{(i)}_{lb} \cap \mathcal{O}^{(j)}_{lb}) = \emptyset, \tag{4.26a}$$

$$\xrightarrow[\text{tivity laws}]{\text{set commuta-}} (\mathcal{O}^{(j)}_{\pi} \cap \mathcal{O}^{(j)}_{lb}) \cap \mathcal{O}^{(i)}_{lb} = \emptyset, \tag{4.26b}$$

$$\mathcal{O}^{(j)}_{\pi} \cap \mathcal{O}^{(i)}_{lb} = \emptyset. \tag{4.26c}$$

According to Equation 4.24a, it has

$$\mathcal{O}^{(i)}_{lb} = \mathcal{O}^{(i)}_{lb} \cup \mathcal{O}^{(i)}_{\pi},$$

which is inserted in Equation 4.26c, and it has

$$\mathcal{O}^{(j)}_{\pi} \cap (\mathcal{O}^{(i)}_{lb} \cup \mathcal{O}^{(i)}_{\pi}) = \emptyset, \tag{4.27a}$$

$$\xrightarrow[\text{tivity laws}]{\text{set distribu-}} (\mathcal{O}^{(j)}_{\pi} \cap \mathcal{O}^{(i)}_{lb}) \cup (\mathcal{O}^{(j)}_{\pi} \cap \mathcal{O}^{(i)}_{\pi}) = \emptyset, \tag{4.27b}$$

$$\xrightarrow{4.26c} \emptyset \cup (\mathcal{O}^{(j)}_{\pi} \cap \mathcal{O}^{(i)}_{\pi}) = \emptyset, \tag{4.27c}$$

$$\mathcal{O}^{(j)}_{\pi} \cap \mathcal{O}^{(i)}_{\pi} = \emptyset. \tag{4.27d}$$

Equation 4.27d indicates that the predicted occupied sets of vehicles i and j is collision-free, i.e., vehicles i and j will never collide in the prediction horizon H_p. □

Based on Proposition 9, two coupled vehicles at merging- or crossing-adjacent lanelets are decoupled if one of them is not allowed to enter their lanelet crossing area. One may ask which vehicle should be forbidden to enter their lanelet crossing area. A straightforward answer would be: Forbid the one with a lower priority. However, there are two other cases. The first one is that the lower-priority one has already entered this area, which can occur due to, for example, the usage of a short prediction

horizon so that vehicles are coupled too late. In this case, the one with a higher priority can be forbidden from entering this area to decouple them, forcing them to swap their priorities. Another case is that both vehicles have already entered this area. In this case, they cannot be decoupled; therefore, the lower-priority vehicle must consider the predicted trajectory of the other one.

The advantage of decoupling is enormous. The coupling graph after the decoupling of vehicles in Fig. 4.14(a) is depicted in Fig. 4.14(b). The number of coupling edges is reduced from eight to two. The remaining two coupling edges are between vehicles at the same lanelet and forking-adjacent lanelets. Other types of lanelets where vehicles cannot be decoupled are longitudinal-, left-, right-adjacent lanelets. In addition, the coupling cycle is disappeared compared with Fig. 4.10(a), and the depth of the resulting DAG in Fig. 4.14(b) is only one, which is far smaller than the depth of the DAG after removing the coupling cycle in Fig. 4.10(a). Finally, a vehicle with fewer coupled vehicles means fewer number of constraints while planning its trajectory, possibly reducing the computation load for some types of trajectory planners.

4.5. Communication

Communications are essential in distributed trajectory planning as some information must be available before some actions are allowed to be executed. For example, lower-priority vehicles must have received the predicted trajectories (or predicted occupied sets) from their coupled vehicles with higher priorities in the same group before planning their trajectories; otherwise, the freeness of collisions cannot be guaranteed.

In the proposed framework, Robot Operating System (ROS) 2 [MFG+22], an open-source robotics middleware suite, is used as a communication interface. A ROS 2 network consists of nodes. Nodes can communicate with each other via messages. Messages can be sent to specific topics published by nodes. A node can publish multiple topics. To receive messages from topics, nodes must subscribe them.

There are two topics: *main* and *sync*. The messages in topic *main* contain

1. the predicted occupied sets (i.e., occupied sets of the predicted trajectories),
2. the ID of the sender,
3. the time step when the message is sent,
4. the predicted lanelets,
5. the predicted trims, and
6. whether the fallback trajectory is used ($isFallback \in \mathbb{B}$).

Henceforth they will be called the *main* messages. In ROS 2 of MATLAB, one can access messages using one property of the ROS 2 publisher, called *LatestMessage*.

As its name says, vehicles can only read the last messages sent by others. If new messages are sent before the last messages have been read, those last messages will be unavailable. The *sync* messages will be used to synchronize the sending and the reading of the *main* messages to avoid this problem. They contain only one information: the current time step of the sender.

A visualization of the ROS 2 network used in this thesis is given in Fig 4.15. Each node corresponds to one vehicle. Each vehicle can read and send messages to two topics. $Send(sync, i)$ indicates that vehicle i sends a message to topic *sync*. If vehicle j wants to read this message, $Read(sync, i)$ will be used. If i is replaced by a set of vehicles, it means reading messages from all those vehicles. For example, $Read(main, \mathcal{V}_{ig}^{(i+)})$ indicates reading the *main* messages from higher-priority vehicles that have in-group couplings with vehicle i.

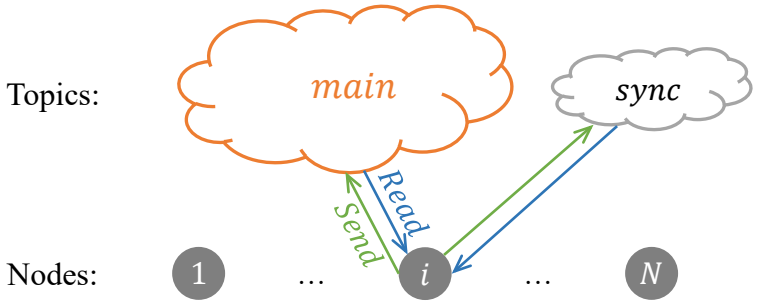


Figure 4.15.: The used ROS 2 network, consisting of N nodes and two topics.

4.6. Overall Algorithm of the framework

This Section summarizes the overall algorithm of the proposed framework.

First, here are some denotations. The set of vehicles that are decoupled (by the vehicle decoupling strategy) with vehicle i is denoted as $\mathcal{V}_d^{(i)}$. The set of vehicles that are still coupled with vehicle i is denoted by $\mathcal{V}^{(i)}$, which contains two subsets according to the types of their couplings: 1) $\mathcal{V}_{ig}^{(i)}$ if they have in-group couplings 2) $\mathcal{V}_{cg}^{(i)}$ if they have cross-group couplings. Subscript $-$ will be used to indicate lower priorities, and $+$ will be used for higher priorities. For example, $\mathcal{V}_{ig}^{(i-)}$ is a set of lower-priority vehicles that have in-group couplings with vehicle i.

The full algorithm of the proposed framework is as follows. All vehicles do the following in parallel (here, it will be described from the perspective of vehicle i):

1. Initialize:

   a) Initialize the ROS 2 network.
   b) $Send(main, i)$ the ID i, the current time step $k = 0$, the predicted lanelets, and the predicted trims.
2. Set $k = 1$ (the first time step).
3. $Read(main, \mathcal{V} \backslash i)$ and wait until all others have sent their *main* messages from the last time step $k - 1$ (this can be checked by looking at the time step when messages are sent).
4. Inform traffic status:
   a) $Read(main, \mathcal{V} \backslash i)$ the predicted lanelets and predicted trims of others, and measure their positions.
   b) $Send(sync, i)$ the current time step k.
   c) Determine the edge-weighted directed acyclic graph (see Section 4.2).
   d) Option: Decouple vehicles wherever possible (see Section 4.4.3).
   e) Group vehicles (see Section 4.3).
5. $Read(main, \mathcal{V}_{ig}^{(i+)})$ and wait for the *main* messages from higher-priority vehicles that have in-group couplings with it.
6. If $\exists j \in \mathcal{V}_{ig}^{(i+)} : isFallback^{(j)} = true$, vehicle i must also use its fallback trajectory: set its trajectory $\pi^{(i)}$ to the fallback trajectory $\pi_{fb}^{(i)}$ and set $isFallback^{(i)}$ to $true$. Otherwise, try to plan a new trajectory $\pi^{(i)}$:
   a) Set its lanelet boundaries as static obstacles.
   b) Option: $\forall j \in \mathcal{V}_{d}^{(i+)}$, do: Set the lanelet crossing area $\mathcal{O}_{lca}^{(i,j)}$ of it and vehicle j as a static obstacle. Vehicle j here is the higher-priority vehicle that is uncoupled with it by the vehicle decoupling strategy. (see Section 4.4.3. This step is only necessary if step 4d is executed).
   c) $\forall j \in \mathcal{V}_{ig}^{(i+)}$, do: Set the predicted occupied sets $\mathcal{O}(\pi_{([k,k+H_p]|k)}^{(j)})$ of vehicle j as dynamic obstacles.
   d) $\forall j \in \mathcal{V}_{ig}^{(i-)} \cap \mathcal{V}_{cg}^{(i-)}$, do: Set the occupied set $\mathcal{O}(\pi_{eb,([k,k+H_p]|k)}^{(j)})$ of the emergency braking trajectory of vehicle j as dynamic obstacles if vehicles i and j have a side-impact collision possibility; otherwise, set the currently occupied set $\mathcal{O}_{c}^{(j)}$ of vehicle j as a static obstacle (see Section 4.4.1).
   e) $Read(main, \mathcal{V}_{cg}^{(i+)})$. $\forall j \in \mathcal{V}_{cg}^{(i+)}$, do:
      i. If the *main* message of vehicle j is from the current time step k, set its predicted occupied sets $\mathcal{O}(\pi_{([k,k+H_p]|k)}^{(j)})$ as dynamic obstacles.
      ii. Otherwise, set its reachable sets $\mathcal{R}_{([k,k+H_p]|k)}^{(j)}$ as dynamic obstacles.
   f) Try to plan a trajectory $\pi^{(i)}$ for H_p time steps $(\pi_{([k,k+H_p]|k)}^{(i)})$ while avoiding all static and dynamic obstacles. If can find one, set $isFallback^{(i)}$ to $false$;

otherwise, set $\pi^{(i)}$ to the fallback trajectory $\pi_{fb}^{(i)}$ and set $isFallback^{(i)}$ to $true$.

7. $Read(sync, \mathcal{V} \backslash i)$ and wait until all others have sent the same k using $sync$ messages.
8. $Send(main, i)$ the occupied set $\mathcal{O}(\pi^{(i)})$ of its trajectory $\pi^{(i)}$, the ID i, the current time step k, the predicted lanelets, the predicted trims, and $isFallback^{(i)}$.
9. $Read(main, \mathcal{V}_{wc}^{(i)})$ and wait until all weakly coupled vehicles with it have sent their $main$ messages. If any of them use their fallback trajectories, set its trajectory $\pi^{(i)}$ also to its fallback trajectory $\pi_{fb}^{(i)}$ and set $isFallback^{(i)} = true$. After that, send the $main$ message again, as described in step 8, and go to step 10.
10. Increment $k = k + 1$ and go to step 3.

Here, all sets refer to the current time step. For example, $\mathcal{V}_{wc}^{(i)}$ is the set of weakly coupled vehicles with vehicle i at the current time step k. A visualization of the overall algorithm is given in Fig. 4.16.

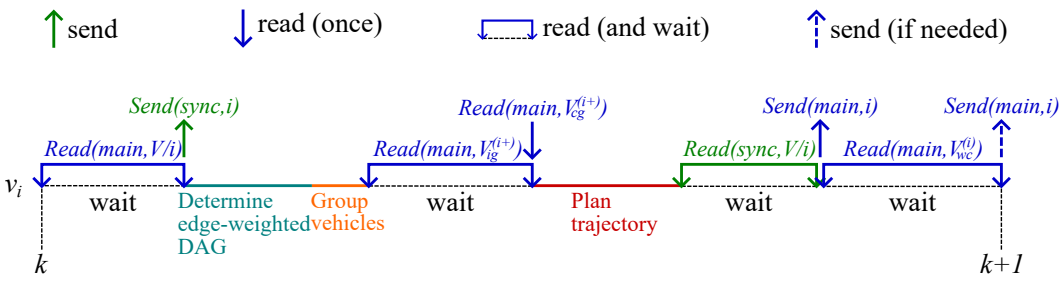


Figure 4.16.: Overall algorithm (from the perspective of vehicle i).

Initialize

At the initial time step $k = 0$, the ROS 2 network will be initialized. Each vehicle sends an initial $main$ message containing its ID, the time step, the predicted lanelets, and the predicted trims. The predicted trims correspond to trim primitives that a vehicle's states will be at if it executes its predicted trajectory. Since no trajectory is planned yet, trims are also not predicted. As each vehicle is assumed initially in a stationary state, the predicted trims will be set to the equilibrium trim. If this is not the case, one can replace them with the actual trim.

Inform the Current Traffic Status

At the beginning of each time step $k > 0$ (step 3), vehicle i reads the *main* messages of others and checks whether the messages come from the last time step. If that is not the case, meaning some vehicles have not finished the planning of the last time step yet, it will wait. After all vehicles have sent their *main* messages from the last time step, vehicle i starts to informs itself of the current traffic status as follows. It reads the predicted lanelets and predicted trims of other vehicles through the *main* messages from the last time step $k-1$ and measures their positions (step 4a). After that, those messages are no longer needed for vehicle i. It sends a *sync* message (see step 4b) to let others know that they can send new *main* messages if they have finished the planning of the current time step k, without overwriting unread old messages. Next, vehicle i determines the edge-weighted directed DAG (step 4c), and its weighting matrix A_w will be used to group vehicles (step 4e) so that it knows its trajectory planning order and which couplings it should consider. Note that step 4d is an optional step. The vehicle decoupling strategy effectively prevents deadlocks at road intersections, which is useful in complex traffic scenarios.

Trajectory Planning

The remaining part (from steps 5 to 9) is about trajectory planning. At step 5, vehicle i reads and waits until higher-priority vehicles that have in-group couplings with it have sent their *main* messages of the current time step k. If one of them triggers a fallback, it must also use its fallback trajectory. Otherwise, it prepares to plan a new trajectory (steps 6a to 6f). First, to not violate its lanelet boundaries, it sets them as static obstacles (step 6a); second, for any vehicles that have been decoupled with it and have higher priorities before decoupling, it sets the lanelet crossing areas with those vehicles as static obstacles (step 6b); third, it sets the predicted occupied sets of higher-priority vehicles that have in-group couplings with it as dynamic (moving) obstacles (step 6c); next, to increase the feasibilities of its coupled vehicles with lower priorities (see Section 4.4.1), it considers either the occupied sets of their emergency braking maneuvers as dynamic obstacles or their currently occupied sets as static obstacles (step 6d); then, to avoid collisions with higher-priority vehicles that have cross-group couplings with it, it considers either their predicted occupied sets (if exist) or reachable sets as dynamic obstacles (step 6e). Finally, it tries to plan a trajectory for H_p time steps while avoiding collisions with all those obstacles (step 6f). If no feasible trajectory can be found, it triggers a fallback and sets its trajectory to its fallback trajectory. Before sending the *main* message (step 8), it waits until others have sent *sync* messages of the current time step (see step 4b), meaning that they have read the old *main* messages from the last time step. After that, it waits until the weakly coupled vehicles with it have also finished their trajectory planning

(step 9). According to the local fallback strategy (see Section 4.4.2), if one of them triggers a fallback, it must also use its fallback trajectory and then sends the *main* message containing the fallback trajectory to overwrite the previously sent *main* message at step 8. At the end, vehicle i goes to the next time step (step 10).

5. Evaluation

The effectiveness and efficiency of the proposed framework are numerically evaluated. The simulations are conducted in MATLAB on a desktop computer with AMD Ryzen 7 5700G with 3.8 GHz processor and 32 GB RAM.

The lanelet-based digital map data comes from the Cyber-Physical Mobility (CPM) Lab, an open-source platform for networked and autonomous vehicles [KSM+21]. The vehicles in the Cyber-Physical Mobility Lab (CPM Lab) are scaled 1:18. The same size is used in the simulations. An overview of the map is given in Fig. 5.1, where the initial positions of all possible vehicles (some simulations use only a part of them) are shown in rectangles. Each vehicle has a predefined reference path (shown in the same color as the rectangle) that requires it to move in a loop. According to the number of vehicles required by each simulation, they are randomly chosen from all 40 vehicles. All previously mentioned types of lanelets are included in the road map of the CPM Lab. It thus provides a wide range of traffic scenarios, such as merging onto the highway, overtaking, and crossing a complex and signal-free intersection.

Table 5.1.: Some parameters.

Parameter	Value	Unit
Vehicle length L	0.22	m
Vehicle width W	0.1	m
Inflation l_{in}	0.01	m
Lanelet width l_W	0.14	m
Sample time T_s	0.2	s

Some geometry parameters of vehicles and lanelets are given in Table 5.1. Each vehicle is approximated by a rectangle with a length of 0.22 m and a width of 0.1 m. Inflation $l_{in} = 0.01$ m is used in both the length and width directions to compensate uncertainties, such as vehicle localization errors and the approximation used in building motion primitives. Note that inflation is only used for collision

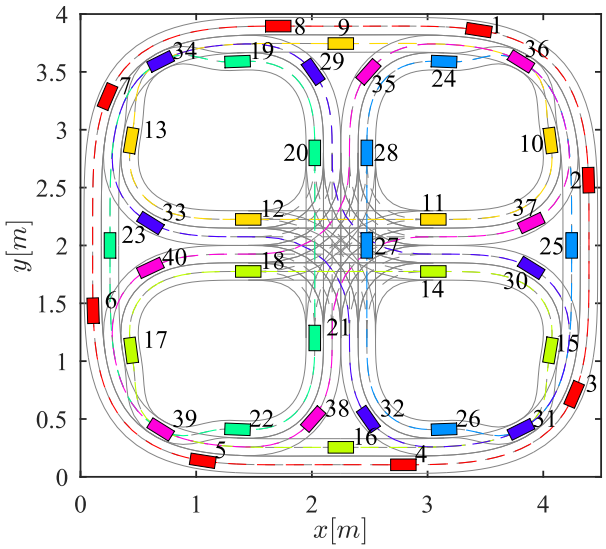


Figure 5.1.: Road map in the CPM Lab. The initial positions of 40 vehicles are shown in rectangles. Vehicles with the same reference path and are drawn in the same color.

checks between vehicles, i.e., collisions between vehicles and lanelet boundaries are checked without inflation. The lanelet width of the CPM Lab is 0.14 m, resulting in a ratio of the vehicle width to the lanelet width of $0.1/0.14 = 71.4\%$ when it drives at a single lanelet. However, this ratio increases if two vehicles move side by side at two parallel lanelets as inflation is used for collision checking between vehicles. In this case, the ratio will be $(0.1 * 1 + 0.2)/(2 * 0.14) = 78.6\%$. The lanelet width in Germany varies from 2.75 to 3.75 m. Normally, it is around 3 m in European. Considering the minimum lanelet width and a car with a width of 2 m (including mirrors) leads to a ratio of 2/2.75=72.7%. Therefore, the lanelets in the CPM Lab are especially narrow. Despite this, the proposed framework performs well, as shown in the following sections.

Section 5.1 evaluates the efficiency of the offline reachability analysis. Section 5.2 evaluates the STAC-based coupling weights and the STAC-based priorities. Section 5.3 evaluates the safety and the control quality of the proposed strategies: the strategy dealing with the prediction inconsistency problem, the feasibility improving strategy, the vehicle decoupling strategy, and the local fallback strategy. Section 5.4 evaluates the real-time capacity of the proposed algorithms.

5.1. Evaluation of the Offline Reachability Analysis

This section evaluates the offline reachability analysis. In Section 4.1.2, two algorithms are proposed to compute offline reachable sets. Algorithm 2 uses a brute-force method, while Algorithm 3 applies the dynamic programming technique.

The prediction horizon H_p is one of the main parameters deciding the computation time of both algorithms. Besides, the complexity and connectivity of trim primitives play key roles. To explore their impacts on the computation time, this section varies the prediction horizon from 1 to 12 and uses two different MPAs, a simpler one from Fig. 4.4 and a more complex one from Fig 2.3. The results are visualized in Fig. 5.2.

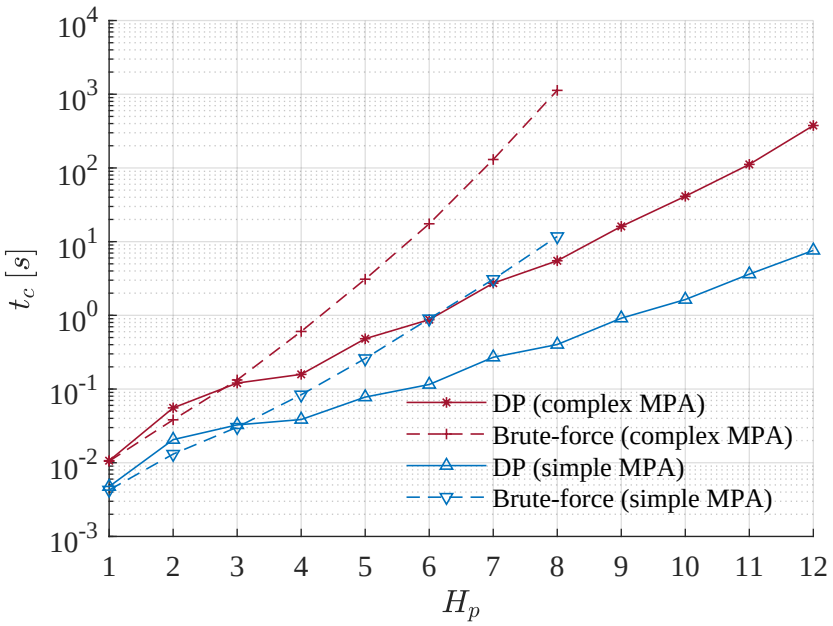


Figure 5.2.: Computation time of the algorithm using dynamic programming (DP) and the brute-force algorithm.

In Fig. 5.2, the computation time of Algorithm 2 is depicted as dashed lines, while that of Algorithm 3 is shown in solid lines. Please note that the y-axis uses a logarithmic scale. Clearly, the computation time grows exponentially for both algorithms as the prediction horizon H_p increases. At $H_p = 8$, Algorithm 2 needs more than one thousand seconds, while Algorithm 3 requires less than 15 s, reducing the computation time by more than 98%. For $H_p > 8$, Algorithm 2 is not evaluated due to its too high computation time.

Next, the computation times for different MPAs are compared. When we compare the red solid line with the blue solid one, we can see that the computation time of Algorithm 3 increases more sharply as the more complex MPA (see 2.3) is used. This also applies to Algorithm 2, where the increasing trend is more obvious. For example, at $H_p = 8$, the computation time of Algorithm 3 is about 0.4 s for the simpler MPA and 5 s for the more complex MPA, which corresponds to an increasing of $(5 - 0.4)/0.4 * 100 = 1150\%$. However, this value for Algorithm 2 is $(1100 - 11)/11 * 100 = 9900\%$ at $H_p = 8$. Interestingly, at $H_p = 2$, Algorithm 2 outperforms Algorithm 3. The reason is that to guarantee the recursive feasibility, the selected trim at the last time step must be an equilibrium trim. That means, in Algorithm 3, when calling Algorithm 2 to compute the reachable sets of the first half prediction horizon, the recursive feasibility is "unnecessarily" guaranteed. One must pay some extra computation time to compute the full reachable sets at time step $H_{p,half}$. Note that both the constraint reachable sets and the full reachable sets at this time step are useful and will be used later. For a detailed explanation see Section 4.1.2. The author does not consider this a worth mentioning downside of Algorithm 3 as in most cases, a prediction horizon of 2 might not be enough and thus will not be chosen. In addition, the computation time of Algorithm 3 is less than 0.1 s here (of course, depending on the complexity of the target trim primitives), which can be nearly neglected. Certainly, one can use Algorithm 2 to calculate the reachable sets if a prediction horizon of 2 is used.

The more complex MPA shown in Fig. 2.3 will be used throughout the remaining evaluation sections.

5.2. Evaluation of Coupling Weights and Priorities

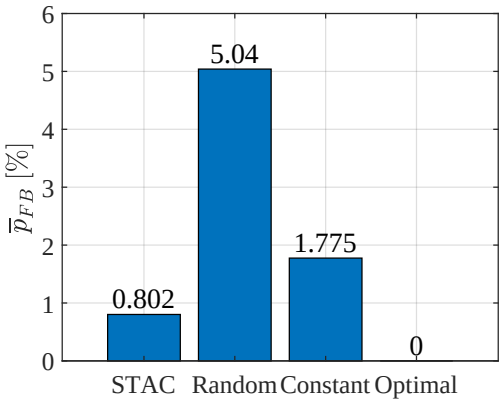
The priorities of vehicles are determined directed once they are grouped (see Section 4.2.6). Inside each group, the priority of each vehicle equals its computation level, e.g., the vehicles in the first computation level have the highest priority and thus plan their trajectories first. Note that the a lower value in priority stands for a higher priority. Before grouping vehicles, one must estimate coupling weights between them. First, Section 5.2.1 evaluates the STAC-based coupling weights. Then, Section 5.2.2 evaluates the STAC-based priorities.

5.2.1. Evaluation of the STAC-Based Coupling Weights

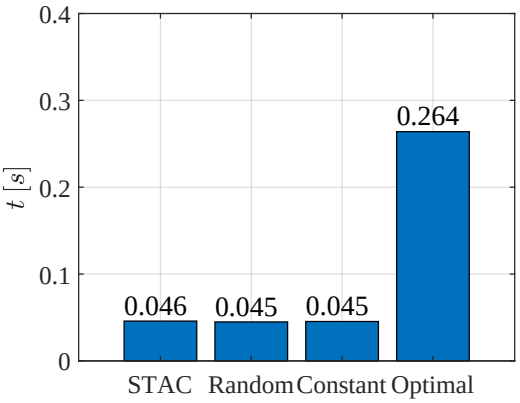
The determination of coupling weights has deciding effect on the grouping results, as the vehicle grouping algorithm groups vehicles based on their coupling weights. Suppose the coupling weights are not appropriately estimated. In that case, strongly coupled vehicles may be divided into different groups, leading to a higher possibility

of infeasibilities as lower-priority vehicles need to avoid the reachable sets of higher-priority vehicles. The proposed STAC-based coupling weights calculated by the weighting function 4.21 will be evaluated in this section. Random coupling weights, constant, and optimal coupling weights are also used for comparison. As their names imply, in the random coupling weights, the coupling weights are assigned randomly from zero to one, while in the constant coupling weights, the coupling weights are constant, 0.5. The optimal coupling weights are introduced in Section 4.2.3.

Totally 18 vehicles are simulated in this section, and a prediction horizon of 5 is used ($H_p = 5$). To test the trueness of the coupling weights representing coupling degrees between vehicles, a purposely low allowed number of computation levels is used, two ($n_{CLs} = 2$). The vehicle decoupling algorithm is turned off to focus on the performance of the vehicle grouping algorithm based on different types of coupling weights. Note that priorities are always determined based on the STAC. Only the way of calculating coupling weights differs. The results are shown in Fig. 5.3.



(b) Average fallback rate.



(c) Average coupling determination time.

Figure 5.3.: Evaluate the STAC-based coupling weights using 18 vehicles.

As vehicles are allowed to use their fallback trajectories when infeasibilities occur, the average fallback rate $\overline{p}_{FB}$ indicates how often vehicles must use their fallback trajectories. The higher the average fallback rate, the more the inconsideration of the current traffic situation. The average fallback rate is calculated by averaging the fallback rate of each time step, which is the percentage of vehicles that use fallback trajectories at a time step. For example, if 5 vehicles among 20 use their fallback trajectories at time step k, the fallback rate of this time step is $p_{FB}^k = 5/20 = 25\%$. The resulting average fallback rate is visualized in Fig 5.3(a). Compared with the constant coupling weights, the STAC-based coupling weights reduce the average fallback rate by almost 55%. The reduction is more than 80% compared with the random coupling weights. If the optimal coupling weights are used to group vehicles,

vehicles will always find feasible trajectories in the simulation, leading to an average fallback rate of zero. However, the computation time is unacceptable if the optimal coupling weights are calculated. As shown in (b), the average coupling determination time (0.264 s) alone is higher than the used sample time (see Table 5.1).

In summary, based on the STAC-based coupling weights that can be calculated rapidly, the proposed vehicle grouping algorithm achieves grouping results nearly as good as the optimal coupling weights, resulting in a largely reduced possibility of infeasibilities.

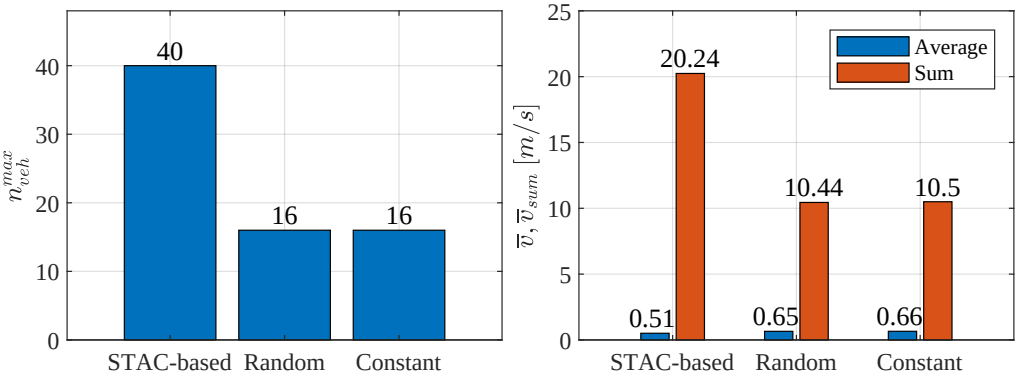
5.2.2. Evaluation of the STAC-Based Priority Assignment Strategy

In this section, the vehicle prioritization algorithm is evaluated. Priority assignment is essential in priority-based trajectory planning. A good prioritization algorithm reduces the possibility of infeasibilities and deadlocks. The proposed prioritization algorithm, i.e., STAC-based priority assignment strategy, is compared with random and constant priority assignment strategies. In the random priority assignment strategy, priorities are assigned randomly. In the constant priority assignment strategy, higher priorities are always given to vehicles with lower IDs. Note that each vehicle has a unique ID in the NCS (see Fig. 5.1). All simulations of this section use a prediction horizon of 5 ($H_p = 5$), while the number of allowed computation levels is set to 4 ($n_{CLs} = 4$).

Use Different Number of vehicles

The maximum controllable number of vehicles n_{veh}^{max} are tested such that all vehicles can at least finish one round without causing any deadlocks. The results are shown in Fig. 5.4(a). The STAC-based priority assignment strategy can let all 40 vehicles run successfully for at least a round, which is two and a half times of that number of the random and the constant priority assignment strategy as their maximum controllable vehicles are the same, 16. This maximum controllable number of vehicles can be understood as the maximum traffic density (the average number of vehicles that occupy a unit length of road space). In Fig. 5.4(b), two kinds of speed are given for each priority assignment strategy: the blue bars for the average speed of all vehicles ($\overline{v}$) and the orange bars for the sum of their average speeds ($\overline{v}_{sum} = \overline{v} \times n_{veh}^{max}$). Note that the sum of the average speeds is an indicator for the traffic capacity (the maximum number of vehicles that can pass a given point in a unit time). The STAC-based priority assignment strategy achieves the highest performance in the sum of average speeds with a value of 20.24 m/s, which is almost twice of that of the random and the constant priority assignment strategy, indicating that the STAC-based priority assignment strategy increases the traffic capacity almost by

100% compared with the other two strategies. A lower value (0.51 m/s) in the average speed of the STAC-based strategy is understandable, as there are far more vehicles driving at an unchanged road space.



(a) Maximum controllable vehicles.

(b) Average speed and sum of speeds.

Figure 5.4.: Evaluate the STAC-based priority assignment strategy using different number of vehicles.

Use the Same number of vehicles

Next, the same number of vehicles is used for the three priority assignment strategies to compare their performance. 16 vehicles are simulated to ensure that all vehicles can run sufficiently long enough (at least one round) without causing deadlocks, which is exactly the maximum controllable vehicles for the random and also the constant priority assignment strategy. The results are presented in Fig. 5.5, where box plots show the average speed of each vehicle, and blue dots depict the total average speeds of all vehicles. For comparison, the free-flow speed is depicted by a blue dashed line. According to [Man10], the free-flow speed for urban streets is defined as "The average running speed of automobiles when they travel along a street under low-volume conditions and when they are not delayed by traffic control devices or other vehicles". In this thesis, the free-flow speed is estimated by averaging the speeds of all vehicles when they plan their trajectories considering only lanelet boundaries but not other vehicles. In this way, the influence of the weakness or strengths of the used trajectory planner can be excluded. The difference between the actual average speed and the free-flow speed indicates the conservativeness of the used control strategy. As one can see, the free-flow speed is around 0.74 m/s. The STAC-based priority assignment strategy achieves 0.676 m/s, which is 91.4% of the free-flow speed, followed by the constant priority assignment strategy, with 88.4% of the free-flow speed. The random priority assignment strategy performs

slightly worse than the constant priority assignment strategy. It is worth noting that in the constant priority assignment strategy, more vehicles have lower average speeds (see its lower whisker). Those are vehicles with higher IDs, as they will always be assigned lower priorities.

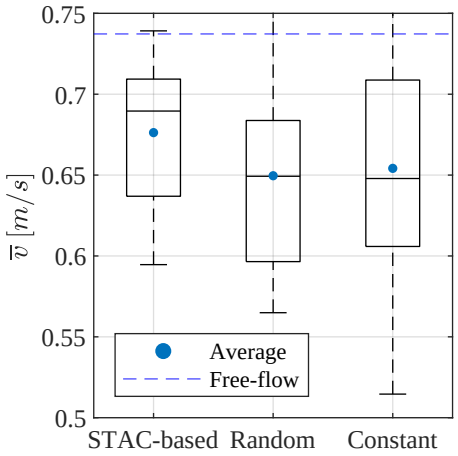


Figure 5.5.: Evaluate the STAC-based priority assignment strategy using 16 vehicles.

In conclusion, the proposed STAC-based priority assignment strategy considerably increases the traffic capacity and the maximum traffic density considerably compared with cases where priorities are assigned randomly or constantly. In addition, it achieves more than 90% of the free-flow speed in a moderate traffic density (16 vehicles).

5.3. Evaluation of Safety and Control Quality

Safety must be guaranteed in NCSs. Section 5.3.1 evaluates the strategy dealing with the prediction inconsistency problem. In Section 5.3.2, the feasibility improving strategy is evaluated. Since deadlocks degrade the control quality largely, the proposed vehicle decoupling strategy preventing deadlocks is also evaluated here. Section 5.3.3 estimates the performance of the proposed local fallback strategy.

5.3.1. Evaluation of the Strategy Dealing with the Prediction Inconsistency Problem

After assigning priorities, the prediction inconsistency problem (see Fig. 1.1(b)) can be addressed by letting lower-priority vehicles consider all the reachable sets of coupled vehicles with higher priorities if they are in different groups. This strategy

is compared with the strategy considering the previous trajectories of their coupled vehicles with higher priorities, which is proposed in [KRSH07]. In concrete, the first trajectory segment of each previous trajectory is discarded as it is already used. The last trajectory segment is repeated to ensure a consistency on the number of prediction horizon. This is exactly the same way as constructing a fallback trajectory (see Equation 4.23). A prediction horizon of 6 is used ($H_p = 6$). The allowed number of computation levels is set to 1 ($n_{CLs} = 1$) to force all vehicles to be in different groups, thus making all couplings cross-group. In this way, the effectiveness of the proposed strategy can be evaluated deeply.

The results are shown in Fig. 5.6. The maximum runtime is used to evaluate the effectiveness of the proposed strategy. As the proposed strategy dealing with the prediction inconsistency problem guarantees the freeness of collisions (see Proposition 8), this maximum runtime refers to the maximum deadlock-free runtime. In the strategy considering the previous trajectories, this runtime refers most often to the maximum collision-free runtime, as collisions are more likely to occur before deadlocks. For each data point in Fig. 5.6, there are three simulations behind, and its final maximum runtime is the average of those three runtimes. The number of vehicles varies from 20 to 30 ($n_{veh} \in [20, 30]$). Therefore, for each strategy, 33 simulations are conducted. In the strategy considering the previous trajectories, only 3 simulations terminate due to deadlocks; that is, collisions occur with a possibility of 91%. In Fig. 5.6, one may notice that the maximum runtime is short if the second strategy is used. However, the proposed strategy increases this runtime notably. For example, the maximum runtime is less than 10 s at $n_{veh} = 24$ in the second strategy, which will be increased to almost 40 s if the proposed strategy is employed. This corresponds to an improvement of 300%. The average improvement among all data points is more than 500%. Note that the vehicle decoupling strategy (see Section 4.4.3) is turned off in this section in order to focus on the target strategy; otherwise, the deadlock-free runtime for the proposed strategy dealing with the prediction inconsistency problem will be so high that it becomes not evaluable. For example, this time is more than 30 min if 20 vehicles are used. In addition, the maximum runtime of the strategy considering the previous trajectories will also be increased noticeably, making its weakness imperceptible. This, however, shows the strength of the vehicle decoupling strategy, which will be evaluated in Section 5.3.2.

5.3.2. Evaluation of the Feasibility Improving Strategy and Vehicle Decoupling Strategy

Section 4.4.1 presented the feasibility improving strategy, and Section 4.4.3 described the vehicle decoupling strategy. Both of the strategies will be evaluated in this section.

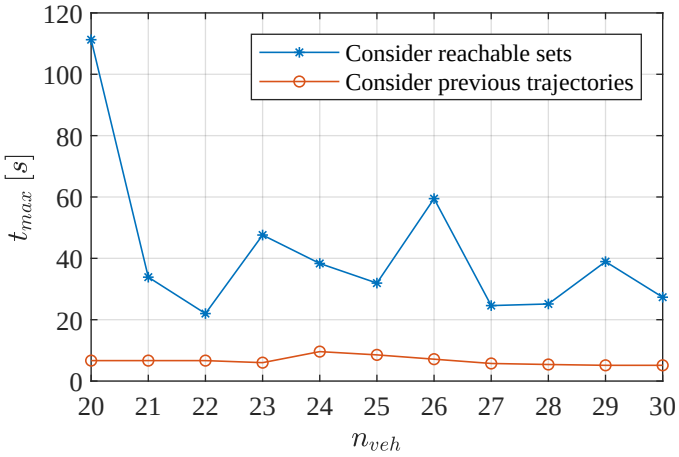


Figure 5.6.: Evaluate the strategy dealing with the prediction inconsistency problem.

The simulations are conducted for four cases:

1. use both strategies,
2. use only the feasibility improving strategy,
3. use only the vehicle decoupling strategy, and
4. do not use any of them.

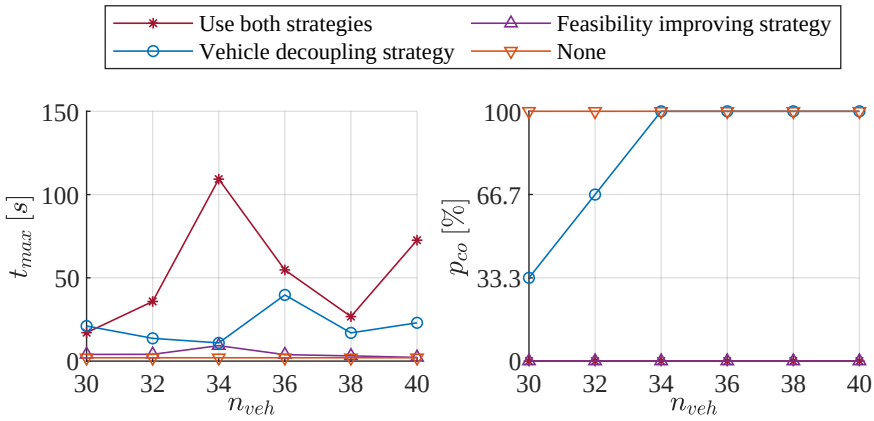
For all cases, a prediction horizon of 5 is used ($H_p = 5$), and the maximum allowed number of computation levels is set to 4 ($n_{CLs} = 4$).

Use Different Number of Vehicles

Firstly, different numbers of vehicles are used to test the maximum runtime and the average fallback rate. High numbers of vehicles (varying from 30 to 40) are used purposely to increase the traffic complexity and thus challenge both strategies. For each specific number of vehicles, vehicles will be randomly selected three times from 40 vehicles (see 5.1). The results will be averaged, i.e., each data point in Fig. 5.7 is the average result of three simulations.

In Fig. 5.7(a), the maximum runtime is visualized. The maximum collision-free runtime is only infinite if the feasibility improving strategy is used (see Proposition 8) because semi-fallbacks are not allowed, i.e., vehicles at a standstill do not need to trigger fallbacks if they cannot find feasible trajectories as they do not endanger others actively (see Section 4.4.2). Therefore, if the feasibility improving strategy is used, the maximum runtime refers to the maximum deadlock-free runtime; otherwise, it could refer to both, depending on whether deadlocks occur earlier or collisions. This information is available in Fig. 5.7(c), where the collision possibility is depicted. As

each data point consists of three simulations, the collision possibility is the percentage of simulations terminated because collisions occur. For example, at $n_{veh} = 32$ and only the vehicle decoupling strategy is used, collisions occur at two out of three simulations, resulting in a collision possibility of 66.7%. It is evident that if the feasibility improving strategy is not used, collisions often occur earlier than deadlocks. Let we look back at Fig. 5.7(a). The vehicle decoupling strategy increases the maximum runtime noticeable. The maximum runtime of using both strategies is more than 20 times of the case where no strategy is used, while this value is around 10 times if only the vehicle decoupling strategy is used, and is about 2 times if only the feasibility improving strategy is employed. Clearly, using both strategies at the same time outperforms the usage of only one of them or none of them.



(a) Maximum runtime. (b) Collision possibility.

Figure 5.7.: Evaluate the feasibility improving strategy and the vehicle decoupling strategy using different number of vehicles.

Use the Same Number of Vehicles

Secondly, 20 vehicles are simulated for all four cases. In Fig. 5.8(a), the average speed of each vehicle is visualized by a box plot, and the average speed of all vehicles is show by a blue dot for each case. Not as expected, the conservativeness of the vehicle decoupling strategy (the third box) is not apparent. Its achieved average speed is about 0.617 m/s, which is around 95.5% of that if none of both strategies are used (0.646 m/s). Compared with the vehicle decoupling strategy, the feasibility improving algorithm (the second box) behaves not conservatively. It even achieves a slightly higher average speed than case 4 (use no strategy). The reason can be seen in Fig. 5.8(b), where the average fallback rate is visualized. Due to its lower fallback rate (2%, which is only a quarter of case 4 with 8%), vehicles are allowed to

plan new trajectories more often based on the current traffic situation. However, the conservativeness is clear if both strategies are combined. The average speed is 0.564 m/s, which is about 87.3% of that of case 4. This means that if two strategies are used at the same time, one has to sacrifice some control quality. One way to handle this is to only use both in complex traffic scenarios.

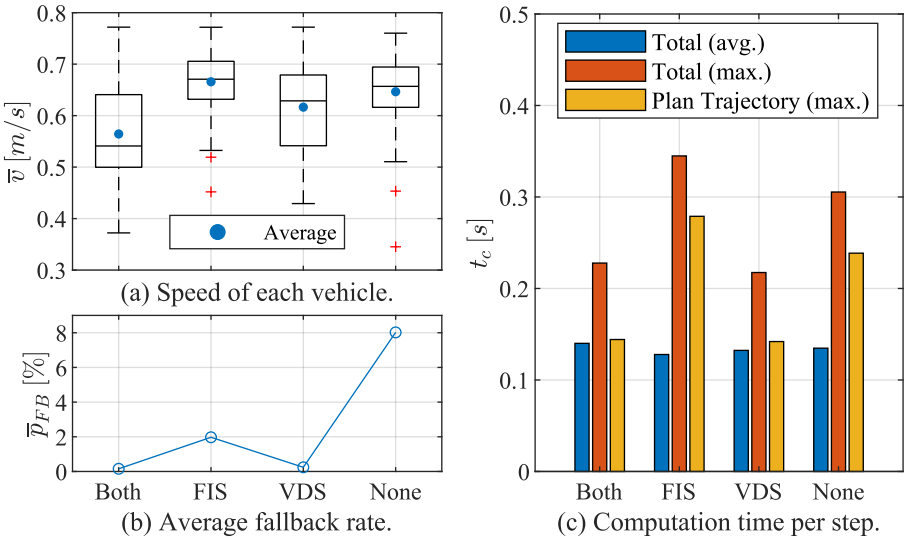


Figure 5.8.: Evaluate the feasibility improving strategy (FIS) and the vehicle decoupling strategy (VDS) using 20 vehicles.

The vehicle decoupling strategy noticeably reduces the maximum trajectory planning time, as shown in Fig. 5.8(c). The average total computation times are almost the same for all the four cases. However, the maximum computation time is more important since the control strategy is only real-time if all actions (see Definition 12) can be finished in the sample time for each time step. One may notice that the maximum computation time is highly related to whether the vehicle decoupling strategy is used. In the first case and the third case, where this strategy is used, the maximum trajectory planning time is the lowest, with a value of about 0.15 s, while this value is more than 0.24 s if it is not used (see the second case and the last case). This corresponds to a decrease of more than 35%. If we compare the last case with the second case, we can see that the maximum trajectory planning time grows from 0.24 to 0.28 s. This is because if the feasibility improving strategy is used, more constraints should be considered when trajectory planning, leading to a higher computation load. From the nearly constant difference between the maximum total computation time and the maximum trajectory planning time, it concludes that the sum of the computation time of other parts, such as coupling determination,

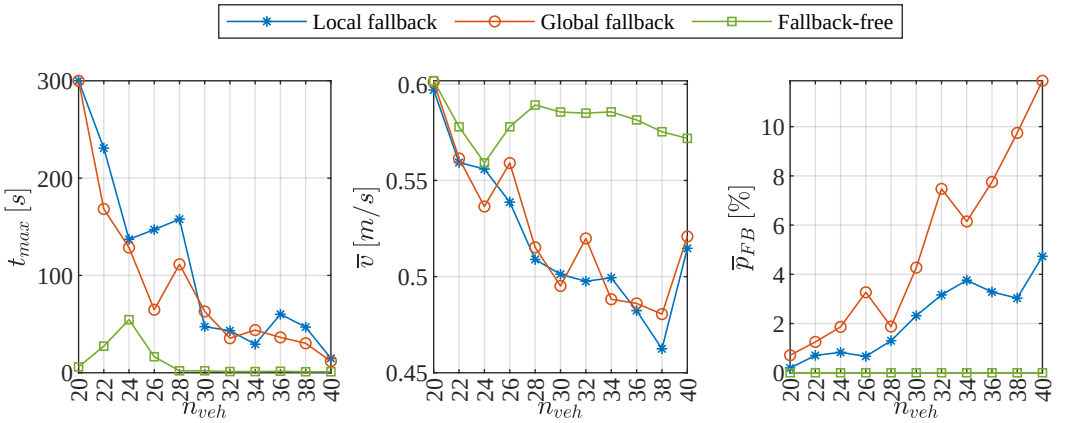
vehicle grouping, and vehicle prioritization, is almost uninfluenced by the usage of the feasibility improving strategy or the vehicle decoupling strategy. That is also proved by the almost unchanged average total computation time. At last, from Fig 5.8(b), we can see that the vehicle decoupling strategy reduces the fallback rate to a great extent.

5.3.3. Evaluation of the Strategy Dealing with Infeasibilities

Vehicles use their fallback trajectories whenever they cannot find feasible trajectories to avoid collisions with other vehicles or lanelet boundaries. Fallback trajectories are constructed based on the planned trajectories of the previous time step (see Equation 4.23). This section evaluates the proposed local fallback strategy (see Section 4.4.2), which will be compared with the global fallback strategy proposed in [Su22]. As a benchmark, the simulation results without fallbacks are also visualized. If fallbacks are not allowed, the simulations will terminate once a vehicle cannot find a feasible trajectory; otherwise, it will collide with other vehicles or its lanelet boundaries. Henceforth, it will be called the fallback-free strategy. A prediction horizon of 6 is used ($H_p = 6$), while the number of computation levels is set to 3 ($n_{CLs} = 3$). The number of vehicles varies from 20 to 40 ($n_{veh} \in [20, 40]$).

In Fig. 5.9, the maximum collision- and deadlock-free runtime (Fig. 5.9(a)), the average speed (Fig. 5.9(b)), and the fallback rate (Fig. 5.9(b)) are visualized. Since the maximum collision-free runtime is infinite for both the local and the global fallback strategies, the maximum runtime in (a) for those two fallback strategies refers to the maximum deadlock-free runtime. The maximum runtime for the fallback-free strategy is the maximum collision-free runtime since here collisions always occur before deadlocks. Note that deadlocks are far better than collisions. High numbers of vehicles are intentionally used in this evaluation section. If only a few vehicles (below 20) are used, the maximum runtime would be very high or even infinite. For example, if 20 vehicles are used, the maximum runtime for the local and the global fallback strategy is more than 300 s. The maximum simulation time is restricted to 300 s to make them evaluable. Once it is reached, the simulations will be terminated.

The maximum runtime in Fig. 5.9(a) refers to the maximum deadlock-free runtime if fallback strategies are used, as the maximum collision-free runtime is infinite here (see Proposition 8). However, this is not the case if fallbacks are not allowed, where collisions will occur once vehicles cannot find feasible trajectories. Therefore, if fallbacks are not allowed, the maximum runtime refers to the maximum collision-free runtime. As one can see in (a), the maximum runtime increases considerably if fallback strategies are used. The local fallback strategy outperforms the global fallback strategy if the number of vehicles is fewer than 30. If this number is exceeded, the performance of both fallback strategies in the maximum runtime differs



(a) Maximum runtime. (b) Average speed of all vehicles. (c) Average fallback rate.

Figure 5.9.: Evaluate the local fallback strategy.

marginally. While it is hard to conclude which fallback strategy leads to a higher average speed (see Fig. 5.9(b)), the local fallback strategy halves the average fallback rate (see Fig. 5.9(c)). Note that if fallbacks are not allowed, the average speed is higher (see Fig. 5.9(b)) because initially, vehicles are not tightly close to each other, where vehicles' speeds are usually high. As the simulations terminate shortly after some time steps (especially if the number of vehicles is more than 26), the resulting average speed is very high. Generally, vehicles have lower speeds at intersections or merging lanelets as they may need to wait for other vehicles. However, such scenarios occur not often at the beginning of the simulations. This is proved at $n_{veh} = 24$, where the average speed of the fallback-free strategy is only slightly higher than the local fallback strategy because the maximum runtime of the fallback-free strategy is not very short (about 50 s).

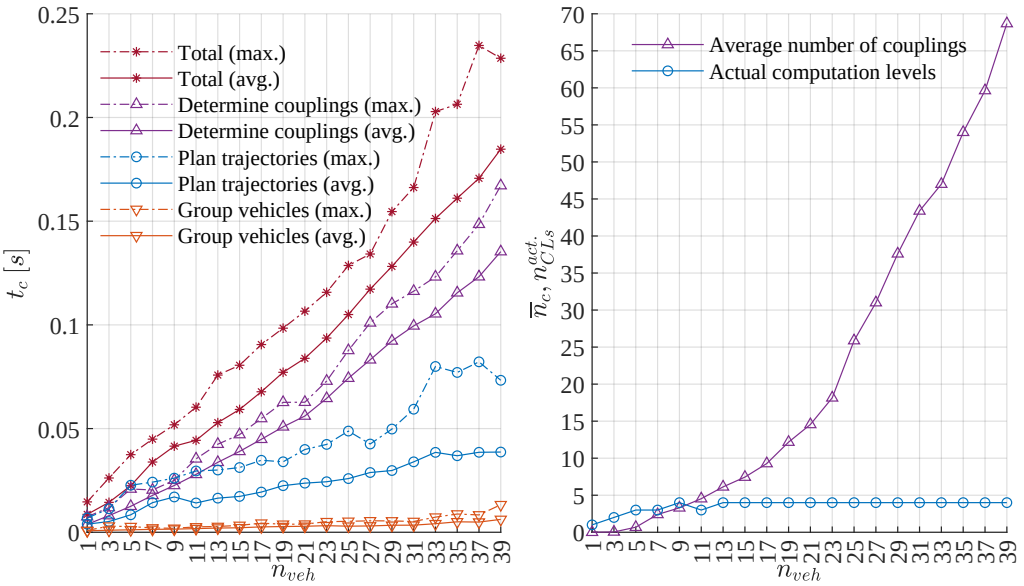
5.4. Evaluation of Real-Time Capacity

A real-time capacity ensures that the control inputs can be calculated on time, which is essential to guarantee collision-free during online operations. Section 5.4.1 evaluates the scalability of the proposed algorithms. Section 5.4.2 estimates the influence of the number of allowed computation levels, which plays a key role in limiting the total trajectory planning time.

5.4.1. Evaluation of the Scalability of Algorithms

Scalability in the sense of the number of vehicles is an essential property of all algorithms used in NCSs. Without good scalability, only a limited number of agents

are allowed to be operated in NCSs. This section evaluates the scalability of the vehicle prioritization algorithm, the coupling determination algorithm, the vehicle grouping algorithm, and the trajectory planning algorithm. A prediction horizon of 5 is used ($H_p = 5$), while the number of computation levels is set to 4 ($n_{CLs} = 4$). The number of vehicles varies from 1 to 39 ($n_{veh} \in [1, 39]$). The results are shown in Fig. 5.10.



(a) Computation time per step.

(b) Couplings and computation levels.

Figure 5.10.: Evaluate the scalability of algorithms.

In Fig. 5.10(a), the maximums of computation time are depicted in dashed lines, while the averages of computation time are shown in solid lines. The total computation time is the sum of the computation time of all parts, mainly consisting of coupling determination time, vehicle grouping time, and trajectory planning time.

At the beginning, the average trajectory planning time increases quickly with the number of vehicles because the actual number of computation levels goes up. It decreases slightly at $n_{veh} = 11$, as the number of the actual computation levels drops by one at this point. Although the number of actual computation levels is unchanged after $n_{CLs} = 11$, the trajectory planning time increases continually. This is because of the increasing number of couplings, as shown by the purple line in Fig. 5.10(b), which results in considering more coupled vehicles during trajectory planning. It may not be noticeable that the average computation time of the coupling determination algorithm and the vehicle grouping algorithm scale quadratically with the number of vehicles. This trend is, however, obvious in their maximum computation times.

Overall, the maximum total computation time grows quadratically while its average time grows almost linear with the number of vehicles.

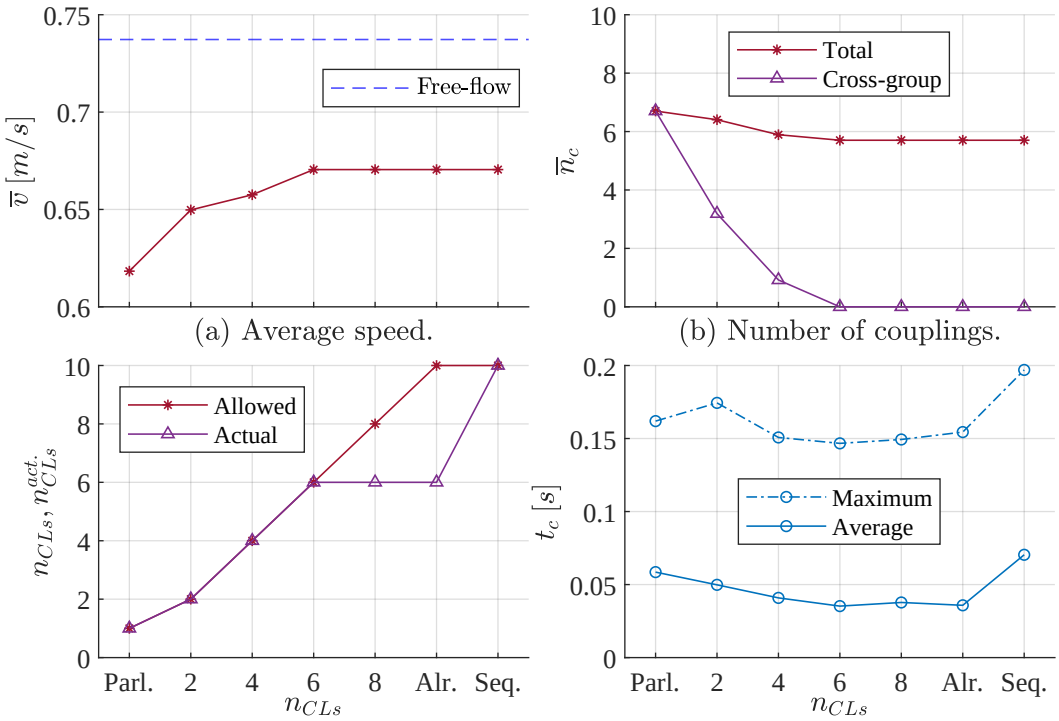
5.4.2. Evaluation of the Allowed Number of Computation Levels

An important parameter of the proposed framework is the allowed number of computation levels n_{CLs}, which is set as a constraint when grouping vehicles and is essential to reduce the actual number of computation levels. This section evaluates the influence of this parameter on the achieved average speed, the resulting average number of couplings, and the total trajectory planning time is evaluated. A prediction horizon of 6 ($H_p = 6$) is used. The vehicle decoupling strategy is turned off to exclude its effect on the control performance. 10 vehicles are selected from 40 vehicles in Fig. 5.1. Those are vehicles with IDs {11,18,19,20,26,27,29,31,32,40}. As one may notice, all those vehicles are about to cross the intersection. In addition, the simulations will be terminated once all vehicles have crossed the intersection. In this way, the final average speed reflects the average speed of crossing the intersection to a large extent.

Four different allowed numbers of computation levels are used ($n_{CLs} \in \{2, 4, 6, 8\}$). As benchmarks, parallel trajectory, sequential trajectory planning, and a state-of-the-art approach from the author B. Alrifaee in his work [Alr17, Sec. 3.5] are compared. A detailed analysis of Alrifaee's strategy is available in Section 3.1

Fig. 5.11(a) visualizes the achieved average speed. It is clear that the vehicles behave most conservatively if they all plan trajectories in parallel. Their average speed is only 0.62 m/s, 83.7% of the free-flow speed. If the allowed number of computation levels becomes 2, the average speed will increase by 5%. It increases continually until the allowed number of computation levels is 6, with a value of 0.67 m/s. After that, the average speed is unchanged, regardless of how many allowed computation levels is set. The reason can be seen in Fig 5.11(b). It shows that the number of cross-group couplings is unchanged after the allowed number of computation levels grows to 6, meaning all couplings are in-group couplings. In this case, vehicles are allowed to plan trajectories in sequence if necessary. No coupling will be considered by reachability analysis. The actual number of computation levels shown in Fig 5.11(c) keeps its value at 6 when $n_{Cls} = 8$, which also explains why increasing the allowed number of computation levels after 6 is not useful. This phenomenon will be called a saturation in the allowed number of computation levels. In sequential trajectory planning, all vehicles plan trajectories one after another, resulting in an actual computation level of 10. However, it achieves the same average speed, 0.67 m/s, but unnecessarily increases the total trajectory planning time.

In Alrifaee's approach, there is only one group of vehicles, and uncoupled and thus parallelizable vehicles will be put in the same computation levels. This strategy reduces the actual computation levels from 10 to 6. If the allowed number of computation levels is set to 6 in this scenario, the proposed framework will achieve exactly the same result as Alrifaee's approach. The advantage of the proposed framework is that it provides a way to further reduce the number of computation levels and to balance the conservativeness of the reachability analysis. For scenarios where real-time capacity is critical, one can set the allowed number of computation levels to a low value; for scenarios where control quality is more important, one can use a higher number of computation levels.



(a) Average speed. (b) Number of couplings.

(c) Computation levels. (d) Trajectory planning time.

Figure 5.11.: Compare different allowed number of computation levels. Three benchmarks: parallel trajectory planning (Parl.), sequential trajectory planning (Seq.), and Alrifaee's approach (Alr.) [Alr17, Sec. 3.5].

One may expect that the total trajectory planning will increase as the number of computation levels grows, which is certainly a factor influencing the total trajectory planning time. However, one must also consider other factors. For example, the lower the allowed number of computation levels, the higher the number of cross-group

couplings (see Fig 5.11(b)). Cross-group couplings will be generally considered by letting lower-priority vehicles consider the reachable sets of higher-priority vehicles, while in in-group couplings, the predicted occupied sets of higher-priority vehicles will be considered. Compared with a predicted occupied set, which consists of six points according to the approximation when constructing motion primitives (see Fig. 2.4), a reachable set is far richer in data points (see fig. 4.3). Therefore, in the computation complexity point of view, the consideration of a reachable set is harder than a predicted occupied set. That is why the total trajectory planning time has on clear relationship with the number of computation levels. It even decreases with the number of computation levels sometimes. There is a trade-off when choosing the allowed number of computation levels. However, the author believes that if an all-time trajectory planning algorithm is used, the total trajectory planning time can be well controlled by the allowed number of computation levels.

6. Conclusion

6.1. Summary

This thesis addressed the prediction inconsistency problem between parallel planning vehicles using reachability analysis. The conservativeness caused by reachability analysis was alleviated by letting highly coupled vehicles plan trajectories in sequence. As a result, an MPC-based framework was proposed to combine sequential and parallel trajectory planning, which provided a way to balance the conservativeness of reachability analysis and the number of computation levels in trajectory planning.

The ultimate goal of this framework is to guarantee safety while preventing deadlocks as much as possible. In addition, the real-time capacity of the control strategy is essential.

Safety: Each vehicle is prioritized based on the shortest time to achieve a collision (STAC), which considers its current position, speed, reference path, maximum acceleration and deceleration capacity. Lower-priority vehicles are expected to avoid collisions with higher-priority vehicles. The prediction inconsistency problem between parallel planning vehicles is addressed by letting lower-priority vehicles avoid the reachable sets of higher-priority vehicles. A vehicle grouping algorithm was proposed to mitigate the conservativeness of reachability analysis, letting highly coupled vehicles in the same group where vehicles plan trajectories in sequence. Vehicles in different groups plan their trajectories in parallel to reduce the total trajectory planning time. For a given allowed number of computation levels, the grouping results merely depend on the coupling weights representing the coupled degrees between vehicles, which must be estimated accurately. This thesis defined a weighting function computing the coupling weights based on the STAC.

Real-time capacity: The number of computation levels of each group is constrained by the vehicle grouping algorithm, which aims to limit the total trajectory planning time. In addition, the computation of the reachable sets of nonlinear dynamic systems is generally time-consuming. An approach exploiting the system invariance property was proposed to divide the online reachability analysis burden into offline computations by calculating local reachable sets offline and using them during online operating.

A feasibility improving strategy was proposed to mitigate the incompleteness of priority-based trajectory planning by means of imposing collision avoidance responsibility also on higher-priority vehicles. To be specific, they should either consider the emergency braking maneuvers or the currently occupied sets of their coupled vehicles with lower priorities, which decreases the infeasibility possibility of lower-priority vehicles. A vehicle decoupling strategy was described, and its function effectively prevents vehicles from blocking road intersections.

The proposed framework was evaluated using a lanelet-based road map, where different traffic scenarios could be represented, such as merging, intersection crossing, and side-by-side moving. The dynamics of vehicles were modeled by the kinematic single-track model. The results showed that the STAC-based priority assignment strategy increased the traffic capacity by almost 100% compared both with the random and the constant priority assignment strategies, and the maximum traffic density was increased by more than 100%. The results also showed that the STAC-based coupling weights could represent the coupling degrees between vehicles accurately. If they were used in the vehicle grouping algorithm, the average fallback rate was decreased by almost 55% compared with constant coupling weights and more than 80% compared with random coupling weights. The usage of them could even achieve grouping results that were nearly as good as the optimal coupling weights. In addition, the prediction inconsistency problem was well addressed. The proposed strategy increased the maximum collision-free runtime to infinite, i.e., vehicles would never collide with each other. Furthermore, the proposed vehicle decoupling strategy and the feasibility improving strategy increased the maximum deadlock-free runtime by more than 20 times. The vehicle decoupling strategy could additionally decrease the trajectory planning time by more than 35% because fewer constraints should be considered when solving the trajectory planning problem. Finally, the proposed framework showed its ability to balance the conservativeness of reachability analysis and the number of computation levels.

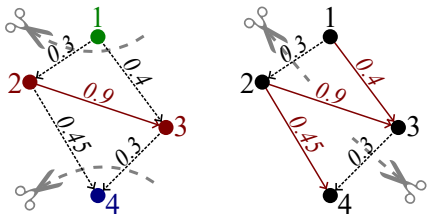
6.2. Future Work

In the future, the author plans to use a high-fidelity MPA to discretize the system states more accurately, thus the control inputs. Backward moving can also be allowed, which is useful when deadlocks occur. One must pay attention to priority assignment if a vehicle is intended to move backwards. One strategy is to give the highest priorities to it so that its back moving trajectory can be considered by others with lower priorities. If necessary, others will also move backwards to avoid collisions with it.

The coupling determination time scales quadratically with the number of vehicles. The main reason is that each vehicle should estimate couplings between all vehicles in NCSs. In the future, the author plans to let each vehicle determine couplings for a set of vehicles, e.g., vehicles around it, making the coupling determination time independent of the total number of vehicles. One must ensure that each vehicle will get the same grouping results; otherwise, two vehicles may mutually think that they should plan trajectories first.

An automatic adaption of the allowed number of computation levels could be developed. According to the efficiency of trajectory planners, the allowed number of computation levels can be automatically adjusted so that the total trajectory planning time is controlled. The author plans to use an all-time trajectory planner, where trajectories will be available once a defined time is reached. In this way, the allowed number of computation levels can be easily adjusted.

Finally, an important future work is to improve the graph partitioning algorithm. The current algorithm (Algorithm 5) cuts vertices into subgraphs such that each subgraph has a lower depth than allowed. If one applies it to the example in Fig. 6.1(a) with the allowed number of computation levels being two, three subgraphs will be obtained, and four edges will be cut. This means that the corresponding four couplings will be considered by reachability analysis. This number can be reduced if one cuts the edges of the target coupling graph such that each subgraph has a lower depth than allowed, as shown in Fig. 6.1(b). Here, only two edges are cut. Therefore, two couplings will be considered by reachability analysis. The resulting number of computation levels is still two while the conservativeness is reduced, e.g., vehicle 3 does not need to avoid the whole reachable set of vehicle 1; instead, it can wait until vehicle 1 has finished its planning.



(a) Cut vertices. (b) Cut edges.

Figure 6.1.: Comparison of cutting vertices and cutting edges.

Bibliography

[ADG07] Asarin, Eugene ; Dang, Thao ; Girard, Antoine: Hybridization Methods for the Analysis of Nonlinear Systems. In: Acta Informatica 43 (2007), Februar, Nr. 7, 451–476. https://doi.org/10.1007/s00236-006-0035-7

[AFG21] Althoff, Matthias ; Frehse, Goran ; Girard, Antoine: Set Propagation Techniques for Reachability Analysis. In: Annual Review of Control, Robotics, and Autonomous Systems 4 (2021), Nr. 1, 369–395. https://doi.org/10.1146/annurev-control-071420-081941

[AGS13] Asselborn, L. ; Gross, D. ; Stursberg, O.: Control of Uncertain Nonlinear Systems Using Ellipsoidal Reachability Calculus. In: IFAC Proceedings Volumes 46 (2013), Januar, Nr. 23, 50–55. https://www.sciencedirect.com/science/article/pii/S1474667016316342

[AHG13] Althoff, Matthias ; Hess, Daniel ; Gambert, Florian: Road Occupancy Prediction of Traffic Participants. In: 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), 2013, S. 99–105

[AK11] Althoff, Matthias ; Krogh, Bruce H.: Zonotope Bundles for the Efficient Computation of Reachable Sets. In: 2011 50th IEEE Conference on Decision and Control and European Control Conference, 2011, S. 6814–6821

[Alr17] Alrifaee, Bassam: Networked Model Predictive Control for Vehicle Collision Avoidance, RWTH Aachen University, Doctor Thesis, 2017

[Alt10] Althoff, Matthias: Reachability Analysis and Its Application to the Safety Assessment of Autonomous Cars, Technische Universität München, Diss., 2010. https://mediatum.ub.tum.de/963752

[Alt13] Althoff, Matthias: Reachability Analysis of Nonlinear Systems Using Conservative Polynomialization and Non-Convex Sets. In: Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control - HSCC '13. Philadelphia, Pennsylvania, USA : ACM Press, 2013, 173

[AM16] Althoff, Matthias ; Magdici, Silvia: Set-Based Prediction of Traffic Participants on Arbitrary Road Networks. In: IEEE Transactions on

Intelligent Vehicles 1 (2016), Juni, Nr. 2, S. 187–202

[ASB08] Althoff, Matthias ; Stursberg, Olaf ; Buss, Martin: Reachability Analysis of Nonlinear Systems with Uncertain Parameters Using Conservative Linearization. In: 2008 47th IEEE Conference on Decision and Control, 2008, S. 4042–4048

[BB08] Bobanac, Vedran ; Bogdan, Stjepan: Routing and Scheduling in Multi-AGV Systems Based on Dynamic Banker Algorithm. In: 2008 16th Mediterranean Conference on Control and Automation, 2008, S. 1168–1173

[BBT02] Bennewitz, Maren ; Burgard, Wolfram ; Thrun, Sebastian: Finding and Optimizing Solvable Priority Schemes for Decoupled Path Planning Techniques for Teams of Mobile Robots. In: Robotics and Autonomous Systems 41 (2002), November, Nr. 2, 89–99. https://www.sciencedirect.com/science/article/pii/S0921889002002567

[BF17] Bashiri, Masoud ; Fleming, Cody H.: A Platoon-Based Intersection Management System for Autonomous Vehicles. In: 2017 IEEE Intelligent Vehicles Symposium (IV), 2017, S. 667–672

[BJF18] Bashiri, Masoud ; Jafarzadeh, Hassan ; Fleming, Cody H.: PAIM: Platoon-based Autonomous Intersection Management. In: 2018 21st International Conference on Intelligent Transportation Systems (ITSC), 2018, S. 374–380

[Bol12] Bollobas, Bela: Graph Theory: An Introductory Course. Springer Science & Business Media, 2012

[Buc89] Buckley, S.J.: Fast Motion Planning for Multiple Moving Robots. In: Proceedings, 1989 International Conference on Robotics and Automation. Scottsdale, AZ, USA : IEEE Comput. Soc. Press, 1989, 322–326

[BZS14] Bender, Philipp ; Ziegler, Julius ; Stiller, Christoph: Lanelets: Efficient Map Representation for Autonomous Driving. In: 2014 IEEE Intelligent Vehicles Symposium Proceedings, 2014, S. 420–425

[CA13] Camacho, Eduardo F. ; Alba, Carlos B.: Model Predictive Control. Springer Science & Business Media, 2013

[CE16] Chen, Lei ; Englund, Cristofer: Cooperative Intersection Management: A Survey. In: IEEE Transactions on Intelligent Transportation Systems 17 (2016), Nr. 2, S. 570–586

[CES71] Coffman, E. G. ; Elphick, M. ; Shoshani, A.: System Deadlocks. In: ACM Computing Surveys 3 (1971), Juni, Nr. 2, 67–78. https://doi.org/10.1145/356586.356588

[CK03] Chutinan, A. ; Krogh, B.H.: Computational Techniques for Hybrid System Verification. In: IEEE Transactions on Automatic Control 48

(2003), Nr. 1, S. 64–75

[ČNKS15] Čáp, Michal ; Novák, Peter ; Kleiner, Alexander ; Selecký, Martin: Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots. In: IEEE Transactions on Automation Science and Engineering 12 (2015), Juli, Nr. 3, S. 835–849

[CNS+13] Čáp, Michal ; Novák, Peter ; Selecký, Martin ; Faigl, Jan ; Vokffnek, Jiff: Asynchronous Decentralized Prioritized Planning for Coordination in Multi-Robot System. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2013, S. 3822–3829

[CWC13] Chu, An-Chiang ; Wu, Bang Y. ; Chao, Kun-Mao: A Linear-Time Algorithm for Finding an Edge-Partition with Max-Min Ratio at Most Two. In: Discrete Applied Mathematics 161 (2013), Mai, Nr. 7, 932–943. https://www.sciencedirect.com/science/article/pii/S0166218X12004337

[DC12] Dunbar, William B. ; Caveney, Derek S.: Distributed Receding Horizon Control of Vehicle Platoons: Stability and String Stability. In: IEEE Transactions on Automatic Control 57 (2012), März, Nr. 3, S. 620–633

[dFS13] de Campos, Gabriel R. ; Falcone, Paolo ; Sjoberg, Jonas: Autonomous Cooperative Driving: A Velocity-Based Negotiation Approach for Intersection Crossing. In: 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013). The Hague, Netherlands : IEEE, Oktober 2013, 1456–1461

[DJP+92] Dahlhaus, E. ; Johnson, D. S. ; Papadimitriou, C. H. ; Seymour, P. D. ; Yannakakis, M.: The Complexity of Multiway Cuts (Extended Abstract). In: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing. New York, NY, USA : Association for Computing Machinery, Juli 1992 (STOC '92), 241–251

[DS16] Dellin, Christopher ; Srinivasa, Siddhartha: A Unifying Formalism for Shortest Path Problems with Expensive Edge Evaluations via Lazy Best-First Search over Paths with Edge Selectors. In: Proceedings of the International Conference on Automated Planning and Scheduling 26 (2016), März, 459–467. https://ojs.aaai.org/index.php/ICAPS/article/view/13788

[EL87] Erdmann, Michael ; Lozano-Pérez, Tomás: On Multiple Moving Objects. In: Algorithmica 2 (1987), November, Nr. 1, 477. https://doi.org/10.1007/BF01840371

[FDF05] Frazzoli, E. ; Dahleh, M.A. ; Feron, E.: Maneuver-Based Motion Planning for Nonlinear Systems with Symmetries. In: IEEE Transactions on Robotics 21 (2005), Nr. 6, S. 1077–1091

[FFed] Ford, L. R. ; Fulkerson, D. R.: Maximal Flow Through a Network. In: Canadian Journal of Mathematics 8 (1956/ed), 399–404. https://www.cambridge.org/core/journals/canadian-journal-of-mathematics/article/maximal-flow-through-a-network/5D6E55D3B06C4F7B1043BC1D82D40764

[FOW19] Flasskamp, Kathrin ; Ober-Blöbaum, Sina ; Worthmann, Karl: Symmetry and Motion Primitives in Model Predictive Control. In: Mathematics of Control, Signals, and Systems 31 (2019), Dezember, Nr. 4, 455–485. http://link.springer.com/10.1007/s00498-019-00246-7

[Fre05] Frehse, Goran: PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. In: Morari, Manfred (Hrsg.) ; Thiele, Lothar (Hrsg.): Hybrid Systems: Computation and Control. Berlin, Heidelberg : Springer, 2005 (Lecture Notes in Computer Science), S. 258–273

[FS12] Farina, Marcello ; Scattolini, Riccardo: Distributed Predictive Control: A Non-Cooperative Algorithm with Neighbor-to-Neighbor Communication for Linear Systems. In: Automatica 48 (2012), Juni, Nr. 6, 1088–1096. https://www.sciencedirect.com/science/article/pii/S000510981200129X

[GA19] Gruber, Felix ; Althoff, Matthias: Scalable Robust Model Predictive Control for Linear Sampled-Data Systems. In: 2019 IEEE 58th Conference on Decision and Control (CDC), 2019, S. 438–444

[GG08] Girard, Antoine ; Guernic, Colas L.: Efficient Reachability Analysis for Linear Systems Using Support Functions. In: IFAC Proceedings Volumes 41 (2008), Januar, Nr. 2, 8966–8971. https://www.sciencedirect.com/science/article/pii/S1474667016403939

[Gir05] Girard, Antoine: Reachability of Uncertain Linear Systems Using Zonotopes. Version: 2005. http://link.springer.com/10.1007/978-3-540-31954-2_19. In: Hutchison, David (Hrsg.) ; Kanade, Takeo (Hrsg.) ; Kittler, Josef (Hrsg.) ; Kleinberg, Jon M. (Hrsg.) ; Mattern, Friedemann (Hrsg.) ; Mitchell, John C. (Hrsg.) ; Naor, Moni (Hrsg.) ; Nierstrasz, Oscar (Hrsg.) ; Pandu Rangan, C. (Hrsg.) ; Steffen, Bernhard (Hrsg.) ; Sudan, Madhu (Hrsg.) ; Terzopoulos, Demetri (Hrsg.) ; Tygar, Dough (Hrsg.) ; Vardi, Moshe Y. (Hrsg.) ; Weikum, Gerhard (Hrsg.) ; Morari, Manfred (Hrsg.) ; Thiele, Lothar (Hrsg.): Hybrid Systems: Computation and Control Bd. 3414. Berlin, Heidelberg : Springer Berlin Heidelberg, 2005, 291–305

[GJ78] Garey, Michael R. ; Johnson, David S.: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, 1978

[GS22] Gholamhosseinian, Ashkan ; Seitz, Jochen: A Comprehensive Survey on Cooperative Intersection Management for Heterogeneous Connected

Vehicles. In: IEEE Access 10 (2022), S. 7937–7972

[Gue09] Guernic, Colas L.: Reachability Analysis of Hybrid Systems with Linear Continuous Dynamics, Université Joseph-Fourier - Grenoble I, Diss., Oktober 2009. https://tel.archives-ouvertes.fr/tel-00422569

[GYH17] Ge, Xiaohua ; Yang, Fuwen ; Han, Qing-Long: Distributed Networked Control Systems: A Brief Overview. In: Information Sciences 380 (2017), Februar, 117–131. https://www.sciencedirect.com/science/article/pii/S0020025515005551

[HAS14] Hess, Daniel ; Althoff, Matthias ; Sattel, Thomas: Formal Verification of Maneuver Automata for Parameterized Motion Primitives. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014, S. 1474–1481

[Hes] Hespanha, Joao P.: An Efficient MATLAB Algorithm for Graph Partitioning.

[HFL+19] Huang, Chao ; Fan, Jiameng ; Li, Wenchao ; Chen, Xin ; Zhu, Qi: ReachNN: Reachability Analysis of Neural-Network Controlled Systems. In: ACM Transactions on Embedded Computing Systems 18 (2019), Oktober, Nr. 5s, 106:1–106:22. https://doi.org/10.1145/3358228

[JMN93] Johnson, Ellis L. ; Mehrotra, Anuj ; Nemhauser, George L.: Min-Cut Clustering. In: Mathematical Programming 62 (1993), Februar, Nr. 1-3, 133–151. http://link.springer.com/10.1007/BF01585164

[KA21a] Kloock, Maximilian ; Alrifaee, Bassam: Synchronization of Local Plans for Cooperative Distributed Decision-Making. https://www.techrxiv.org/articles/preprint/Synchronization_of_Local_Plans_for_Cooperative_Distributed_Decision-Making/16622017/1. Version: September 2021

[KA21b] Kochdumper, Niklas ; Althoff, Matthias: Sparse Polynomial Zonotopes: A Novel Set Representation for Reachability Analysis. In: IEEE Transactions on Automatic Control 66 (2021), September, Nr. 9, S. 4043–4058

[KKM+19] Kloock, Maximilian ; Kragl, Ludwig ; Maczijewski, Janis ; Alrifaee, Bassam ; Kowalewski, Stefan: Distributed Model Predictive Pose Control of Multiple Nonholonomic Vehicles. In: 2019 IEEE Intelligent Vehicles Symposium (IV). Paris, France : IEEE, Juni 2019, 1620–1625

[KPBB11] Kalinovcic, Luka ; Petrovic, Tamara ; Bogdan, Stjepan ; Bobanac, Vedran: Modified Banker's Algorithm for Scheduling in Multi-AGV Systems. In: 2011 IEEE International Conference on Automation Science and Engineering, 2011, S. 351–356

[KRSH06] Kuwata, Y. ; Richards, A. ; Schouwenaars, T. ; How, J.P.: Decentralized Robust Receding Horizon Control for Multi-Vehicle Guidance. In: 2006 American Control Conference, 2006, S. 6 pp.–

[KRSH07] Kuwata, Yoshiaki ; Richards, Arthur ; Schouwenaars, Tom ; How, Jonathan P.: Distributed Robust Receding Horizon Control for Multivehicle Guidance. In: IEEE Transactions on Control Systems Technology 15 (2007), Juli, Nr. 4, S. 627–641

[KSM+19] Kloock, Maximilian ; Scheffe, Patrick ; Marquardt, Sascha ; Maczijewski, Janis ; Alrifaee, Bassam ; Kowalewski, Stefan: Distributed Model Predictive Intersection Control of Multiple Vehicles. In: 2019 IEEE Intelligent Transportation Systems Conference (ITSC). Auckland, New Zealand : IEEE, Oktober 2019, 1735–1740

[KSM+21] Kloock, Maximilian ; Scheffe, Patrick ; Maczijewski, Janis ; Kampmann, Alexandru ; Mokhtarian, Armin ; Kowalewski, Stefan ; Alrifaee, Bassam: Cyber-Physical Mobility Lab: An Open-Source Platform for Networked and Autonomous Vehicles. In: arXiv:2004.10063 [cs] (2021), April. http://arxiv.org/abs/2004.10063

[KV07] Kurzhanskiy, Alex A. ; Varaiya, Pravin: Ellipsoidal Techniques for Reachability Analysis of Discrete-Time Linear Systems. In: IEEE Transactions on Automatic Control 52 (2007), Nr. 1, S. 26–38

[LCS16] Luo, Wenhao ; Chakraborty, Nilanjan ; Sycara, Katia: Distributed Dynamic Priority Assignment and Motion Planning for Multiple Mobile Robots with Kinodynamic Constraints. In: 2016 American Control Conference (ACC). Boston, MA, USA : IEEE, Juli 2016, 148–154

[LRF98] Lawley, Mark ; Reveliotis, Spyros ; Ferreira, Placid: The Application and Evaluation of Banker's Algorithm for Deadlock-Free Buffer Space Allocation in Flexible Manufacturing Systems. (1998), S. 28

[LRX21] Li, Juncheng ; Ran, Maopeng ; Xie, Lihua: Efficient Trajectory Planning for Multiple Non-Holonomic Mobile Robots via Prioritized Trajectory Optimization. In: IEEE Robotics and Automation Letters 6 (2021), April, Nr. 2, S. 405–412

[Man10] Manual, Highway C.: HCM2010. In: Transportation Research Board, National Research Council, Washington, DC 1207 (2010)

[May14] Mayne, David Q.: Model Predictive Control: Recent Developments and Future Promise. In: Automatica 50 (2014), Dezember, Nr. 12, 2967–2986. https://www.sciencedirect.com/science/article/pii/S0005109814005160

[MFG+22] Macenski, Steven ; Foote, Tully ; Gerkey, Brian ; Lalancette, Chris ; Woodall, William: Robot Operating System 2: Design, Archi-

tecture, and Uses in the Wild. In: Science Robotics 7 (2022), Mai, Nr. 66, eabm6074. https://www.science.org/doi/10.1126/scirobotics.abm6074

[MSS18] Mandalika, Aditya ; Salzman, Oren ; Srinivasa, Siddhartha: Lazy Receding Horizon A* for Efficient Path Planning in Graphs with Expensive-to-Evaluate Edges. In: Proceedings of the International Conference on Automated Planning and Scheduling 28 (2018), Juni, 476–484. https://ojs.aaai.org/index.php/ICAPS/article/view/13931

[MTHH19] Mirheli, Amir ; Tajalli, Mehrdad ; Hajibabai, Leila ; Hajbabaie, Ali: A Consensus-Based Distributed Trajectory Control in a Signal-Free Intersection. In: Transportation Research Part C: Emerging Technologies 100 (2019), März, 161–176. https://www.sciencedirect.com/science/article/pii/S0968090X18311343

[PA21] Pek, Christian ; Althoff, Matthias: Fail-Safe Motion Planning for Online Verification of Autonomous Vehicles Using Convex Optimization. In: IEEE Transactions on Robotics 37 (2021), Juni, Nr. 3, 798–814. https://ieeexplore.ieee.org/document/9302873/

[PBL+19] Perronnet, Florent ; Buisson, Jocelyn ; Lombard, Alexandre ; Abbas-Turki, Abdeljalil ; Ahmane, Mourad ; El Moudni, Abdellah: Deadlock Prevention of Self-Driving Vehicles in a Network of Intersections. In: IEEE Transactions on Intelligent Transportation Systems 20 (2019), Nr. 11, S. 4219–4233

[Raj11] Rajamani, Rajesh: Vehicle Dynamics and Control. Springer Science & Business Media, 2011

[RSW04] Richards, Nathan ; Sharma, Manu ; Ward, David: A Hybrid A*/Automaton Approach to On-Line Path Planning with Obstacle Avoidance. In: AIAA 1st Intelligent Systems Technical Conference. Chicago, Illinois : American Institute of Aeronautics and Astronautics, September 2004

[SB94] Schoute, Albert L. ; Bouwens, Peter J.: Deadlock-Free Traffic Control with Geometrical Critical Sections. In: Computing Science in the Netherlands, CSN 1994: Jaarbeurs Utrecht 21 En 22 November 1994, Stichting Mathematisch Centrum, Januar 1994, 260–270

[Sha81] Sharir, M.: A Strong-Connectivity Algorithm and Its Applications in Data Flow Analysis. In: Computers & Mathematics with Applications 7 (1981), Januar, Nr. 1, 67–72. https://www.sciencedirect.com/science/article/pii/0898122181900080

[SK03] Stursberg, Olaf ; Krogh, Bruce H.: Efficient Representation and Computation of Reachable Sets for Hybrid Systems. In: Maler, Oded (Hrsg.) ; Pnueli, Amir (Hrsg.): Hybrid Systems: Computation and

Control. Berlin, Heidelberg : Springer, 2003 (Lecture Notes in Computer Science), S. 482–497

[SKA22] Scheffe, Patrick ; Kahle, Julius ; Alrifaee, Bassam: Reducing Computation Time with Priority Assignment in Distributed MPC. https://www.techrxiv.org/articles/preprint/Reducing_Computation_Time_with_Priority_Assignment_in_Distributed_MPC/20304015/1. Version: Juli 2022

[SPFA21] Scheffe, Patrick ; Pedrosa, Matheus Vitor de A. ; Flasskamp, Kathrin ; Alrifaee, Bassam: Receding Horizon Control Using Graph Search for Multi-Agent Trajectory Planning. https://www.techrxiv.org/articles/preprint/Receding_Horizon_Control_Using_Graph_Search_for_Multi-Agent_Trajectory_Planning/16621963/1. Version: September 2021

[Su22] Su, Bingyi: Dynamic Priorities to Increase the Feasibility in Graph-Based Trajectory Planning for Networked Vehicles, RWTH Aachen University, Diplomarbeit, April 2022

[SW97] Stoer, Mechthild ; Wagner, Frank: A Simple Min-Cut Algorithm. In: Journal of the ACM 44 (1997), Juli, Nr. 4, 585–591. https://doi.org/10.1145/263867.263872

[Tar72] Tarjan, Robert: Depth-First Search and Linear Graph Algorithms. In: SIAM Journal on Computing (1972), Juni. https://epubs.siam.org/doi/10.1137/0201010

[VSS10] Velagapudi, Prasanna ; Sycara, Katia ; Scerri, Paul: Decentralized Prioritized Planning in Large Multirobot Teams. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010, S. 4603–4609

[WZ07] Wu, NaiQi ; Zhou, MengChu: Shortest Routing of Bidirectional Automated Guided Vehicles Avoiding Deadlock and Blocking. In: IEEE/ASME Transactions on Mechatronics 12 (2007), Nr. 1, S. 63–72

[ZBS+14] Ziegler, Julius ; Bender, Philipp ; Schreiber, Markus ; Lategahn, Henning ; Strauss, Tobias ; Stiller, Christoph ; Dang, Thao ; Franke, Uwe ; Appenrodt, Nils ; Keller, Christoph G. ; Kaus, Eberhard ; Herrtwich, Ralf G. ; Rabe, Clemens ; Pfeiffer, David ; Lindner, Frank ; Stein, Fridtjof ; Erbs, Friedrich ; Enzweiler, Markus ; Knöppel, Carsten ; Hipp, Jochen ; Haueis, Martin ; Trepte, Maximilian ; Brenk, Carsten ; Tamke, Andreas ; Ghanaat, Mohammad ; Braun, Markus ; Joos, Armin ; Fritz, Hans ; Mock, Horst ; Hein, Martin ; Zeeb, Eberhard: Making Bertha Drive—An Autonomous Journey on a Historic Route. In: IEEE Intelligent Transportation Systems

Magazine 6 (2014), Nr. 2, S. 8–20

[ZHG+20] Zhang, Xian-Ming ; Han, Qing-Long ; Ge, Xiaohua ; Ding, Derui ; Ding, Lei ; Yue, Dong ; Peng, Chen: Networked Control Systems: A Survey of Trends and Techniques. In: IEEE/CAA Journal of Automatica Sinica 7 (2020), Januar, Nr. 1, S. 1–17

A. Appendix

A video demonstration of the proposed MPC-based framework is available here: https://www.youtube.com/watch?v=1LUDNEKFImg

In the video, 30 vehicles are simulated and run in the CPM Lab scenario for 30 seconds. A prediction horizon of 6 is used ($H_p = 6$), and the allowed number of computation levels is set to 3 ($n_{CLs} = 3$). Collision-free are guaranteed, and deadlocks are avoided. Couplings between vehicles are visualized using arrows. Each arrow points from the higher-priority vehicle to the lower-priority vehicle. In-group couplings are depicted in black solid arrows while cross-group couplings are shown in dashed arrows. Couplings that are decoupled by the vehicle decoupling strategy are shown in grey solid arrows. The priorities of vehicles can be judged from their colors. The numbers near vehicles are their IDs. Their reference trajectories are depicted in dots while the predicted trajectories are shown in curves.