CHALMERS
UNIVERSITY OF TECHNOLOGY

Traffic Planning and Coordination of Automated Vehicles

Bachelor's thesis in Computer Science and Engineering

THOMAS ALEXANDERSSON
FARZAD BESHARATI
JOHANNES GUSTAVSSON
MARTIN HILGENDORF
IVAN LYESNUKHIN
JIAN SHIN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Bachelor's thesis DATX02-20-91

Traffic Planning and Coordination of Automated Vehicles

THOMAS ALEXANDERSSON
FARZAD BESHARATI
JOHANNES GUSTAVSSON
MARTIN HILGENDORF
IVAN LYESNUKHIN
JIAN SHIN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2020

Traffic Planning and Coordination of Automated Vehicles
THOMAS ALEXANDERSSON
FARZAD BESHARATI
JOHANNES GUSTAVSSON
MARTIN HILGENDORF
IVAN LYESNUKHIN
JIAN SHIN

Supervisor: Elad Michael Schiller, Associate professor, Department of Computer Science and Engineering (Networks and Systems)
Examiner: Marina Papatriantafilou, Associate professor in the Networks and Systems division, Department of Computer Science and Engineering,

Bachelors's Thesis DATX02-20-91
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

Abstract

Systems for autonomous vehicles can increase productivity within the transport industry. Automated driving systems will liberate people from the need to drive, allowing the driver to be engaged with other vocational activities. Moreover, they will allow people and products that cannot drive to travel without the need to hire driver services. In order for such a system to work, a solution that can guarantee efficiency and safety is required. A proposal for a coherent system for planning and coordinating automated vehicles is made. This is accomplished by using *multi-agent path finding* to plan optimal routes for all vehicles in a closed-off road system, and coordinating their journeys in a safe manner with the help of a *collaborative driving management* system. The designed system consists of a continuous conflict-based search algorithm that can plan collision-free paths for up to 100 vehicles simultaneously. The virtual traffic light proposed uses an agreement system where it collaborates with agents to determine paths that should be given the green light. By also designing a service allowing vehicles to connect to the virtual traffic light server corresponding to an upcoming intersection, a safe and optimal solution is achieved. The proposed system can act as a holistic pilot system for autonomous vehicles in closed-off environments such as airports, ports, and mines.

Keywords: Autonomous Driving, Virtual Traffic Lights, Conflict Based Search, Multi-Agent Path finding

Sammanfattning

System för autonoma fordon kan öka produktiviteten inom transportindustrin. Autonoma körsystem befriar människor från att behöva köra, och ger föraren möjligheten att föreha andra arbetsuppgifter. Dessutom låter de personer och produkter som inte kan köra att transporteras utan att behöva anlita förare. För att ett sådant system ska fungera krävs en lösning som kan garantera effektivitet och säkerhet. Här görs ett förslag för ett sammanhängande system för att planera och koordinera autonoma fordon. Detta kan åstadkommas genom användning av ruttplanering för flera agenter för att planera optimala rutter för alla fordon i ett avgränsat vägsystem, samt koordinera deras resor på ett säkert sätt med ett *Collaborative Driving Management* system. Det designade systemet består av en *Conflict Based Search* algoritm som kan planera kollisionsfria rutter för upp till 100 fordon samtidigt. Det virtuella trafikljuset som föreslås använder ett överrenskommelsesystem där det samarbetar med agenter för att hitta vilka vägar som ska få grönt. Detta tillsammans med ett förbindelse-system som låter fordon koppla upp sig mot det virtuella trafikljusets server som motsvarar kommande kritiska sektion, ger oss en säker och optimal lösning. Det föreslagna systemet kan agera som ett holistiskt pilotsystem för autonoma fordon i avgränsade miljöer såsom flygplatser, hamnar, och gruvor.

Nyckelord: Autonom körning, Virtuella trafikljus, Conflict Based Search, Ruttplanering för flera agenter

Acknowledgements

We would like to express our gratitude to Elad Michael Schiller, our supervisor, who introduced us to the concepts used in this project and provided his help to us along the project's development.

We also would like to thank Burkin Günke, the lab responsible, who helped us with technical equipment and theoretical explanation related to the existing code.

We also want to express our gratitude to Chalmers University of Technology for giving us the opportunity to work with this project and providing the technical equipment to use.

A special thanks also to groups of previous years whose code provided us with a base to begin from, as well as this years Group 39, who we have shared the lab resources with.

Nomenclature

A* Pronounced A-Star, single-agent path finding algorithm.

Agent Any type of vehicle in the path planner system.

CBS Conflict Based Search, a form of MAPF.

CCBS Continuous Conflict Based Search, a form of MAPF that uses continuous time.

CDM Collaborative Driving Management.

Critical section A section on the map where vehicles can cross each others path.

Edge A directional pathway connecting nodes to other nodes.

Map Overview of the road network.

MAPF Multi-Agent Path Finding, a problem type wherein multiple agents wish to move from their start positions to their goal positions.

Node A point on a road network graph that is connected to other nodes using edges.

Road Network Graph A directed multigraph used to model the road system.

SIPP Safe Interval Path Planning, variant of A* adapted to use continuous data.

VTL Virtual traffic lights.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

There are currently over 100 billion-dollar investments in the development of automated driving systems and their applications [1]. It is believed that such a system will dramatically change the work-market as well as the productivity of many sectors of the industry. Automation can be particularly useful in industrial zones due to the heavy amount of vehicular traffic, often with dozens of trucks moving around at the same time, and can reduce the number of vehicular accidents.

According to the U.S. Department of Transportation, the top reason for motor vehicle accidents is driver critical errors [2]. It is not uncommon for truck drivers to drive very long distances without taking breaks [3]. With autonomous trucks, the human factor in accidents could be drastically reduced, as well as eliminating the time restrictions currently set by law. As an example, in the EU, a driver is at most allowed to drive for nine hours a day with a total resting time of 45 minutes [4].

Wadud *et al.* write that the carbon footprint of automated vehicles will impact greenhouse gas emissions in several ways [5]. Their research shows that a higher level of automation may lead to reduced energy consumption and greenhouse gas emissions by congestion mitigation, automated eco-driving, and reduced focus on acceleration time.

While the development of autonomous trucks has been ongoing for many years, it was only recently that such a truck managed to perform a successful commercial

cross-country trip in the U.S. [6]. The technology readiness level still has a long way to go, and this project will investigate developing a prototype system for planning and coordinating autonomous traffic.

1.2 Purpose

This work aims to design, develop and evaluate a prototype system that can plan and execute the traversal of multiple vehicles on a road network within industrialized zones, such as airports, ports, or mines, to name a few.

For multiple vehicles to get from point A to point B safely and quickly, the use of *multi-agent path finding* (MAPF) algorithms is needed. The goal is to integrate a MAPF algorithm into the project so that it can meet the requirements.

The safe execution of a plan depends on the vehicles' ability to coordinate the crossing of intersections, roundabouts, narrow roads, bridges, tunnels, etc. To avoid dangerous situations in the form of collisions between the vehicles, the focus will be on implementing collaborative driving management in the critical areas.

1.3 Challenges

The main challenge is to route and coordinate multiple vehicles in a road environment, and to investigate how a system for this could work. To create the desired system, challenges in three main areas must be overcome: optimal route planning for multiple agents using a MAPF algorithm, coordination of traffic in critical sections by implementing collaborative driving management, and simulation of the overall system to evaluate its behavior.

A large part of the challenge comes in properly routing the vehicles before they leave for their destinations, to generate and verify routes that are collision-free from start to end.

Every road system consists of intersections, where the movement needs to be well-coordinated and safely executed by every traffic participant to avoid collisions between the road users. In dangerous situations, many vehicles cannot abruptly brake or stop the movement, crossing through the risk road region. Therefore, the implementation of safe coordination in the intersections becomes critical. The challenge is to create a system that is designed to handle multiple types of intersections, and that will help vehicles traverse through the risk regions without collisions by coordinating their movement while inside the dangerous zones.

The safety criteria used in this project, forbid the entrance of more than one vehicle to critical sections of the road, such as the same zone in an intersection. This challenge is demanding such that its safety must never be violated and the always-available service has to be scalable in the number of vehicles that it can concurrently service.

1.4 Limitations

Since the prospective automated driving system will have to follow the rules, the scope of this project will assume that all vehicles follow the traffic rules under all circumstances. This drastically reduces the complexity of the decision-making in the individual agents, allowing the team to focus more on developing the path finding- and coordination systems. Vehicles not in the system (rogue vehicles), temporarily/dynamically closed-off roads, and other external factors that could affect traffic, such as weather, are also not accounted for.

The original project scope stated that the implemented solution will need to run in the available lab environment containing a scaled model of a closed road system with model vehicles, but due to COVID-19 and the decision to work from home, this was no longer possible.

1.5 Related Work

The foundations of the tools used in this project originate from previous bachelor thesis projects in this domain that have been carried out at Chalmers. The road layout of the physical testbed and code for autonomous driving of the model truck based on a kinematic model has been reused, as well as the use of the Robot Operating System (ROS) to implement the distributed system.

The CDM implementation originates from the implementation of a virtual traffic light from 2019, but improves and extends upon its design to incorporate a multitude of new intersection types [7].

Andréasson *et al.* present in their master's thesis a system for safe handling and coordinating cars in intersections that can detect system and communication failures as well as message corruption [8]. The authors propose a solution, which includes a system for entering and exiting critical intersections via a join and departure zone which have been an inspiration for our work.

Casimiro *et al.* in "Self-Stabilizing Manoeuvre Negotiation" propose the design of

a safe and cost-efficient automated driving system by using self-stabilizing algorithms. The given algorithms provide a solution on a safe maneuver traffic execution even in combination with problematic conditions such as inaccurate sensory information and malicious behavior.

"Conflict-Based Search for Optimal Multi-Agent Pathfinding" introduces a new algorithm proposed to solve multi-agent path finding problems on maps such as road networks more efficiently while still keeping an optimal solution [10]. This algorithm is the main inspiration for the path planner implemented in this project and is explained in Section 2.1.1.

In "SIPP: Safe Interval Path Planning for Dynamic Environments", Phillips *et al.* suggest modifications on the popular A* path planning algorithm by grouping time intervals where collisions will occur, while also grouping together time intervals that are collision-free [11]. This greatly reduces execution time compared to regular A*.

Andreychuk *et al.* in "Multi-Agent Pathfinding with Continuous Time" improves upon the previous work by Sharon *et al.* by using a modified SIPP algorithm together with CBS. The implementation also takes into account the geometric shape of the agents and obstacles occupying the map allowing a real-world use. This version of CBS, called CCBS, is the one implemented in this project [12].

Other work in the area of vehicular coordination also includes [13]–[21].

1.6 Our Contribution

This work introduces an architecture for a system that provides both optimality and safety for autonomous vehicles traveling in a closed road system.

A continuous MAPF algorithm, known as Continuous Conflict Based Search (CCBS) was implemented. Being continuous allows CCBS to account for more complex aspects of the vehicles involved, such as their top speed and acceleration. This allows for finding fast and collision-free routes for all connected vehicles. The CCBS also takes into account vehicle sizes as geometric shapes when finding routes.

To guarantee safety, the system coordinates and directs traffic in dangerous parts of the road network. The design has been generalized and modularized to support several types of intersections and different traffic directions. This creates a base for a potential extension of the project and, by adding more critical areas and

the possibilities to define more specific characteristics of the intersections, it can handle more traffic situations. To locate what server to talk to when getting close to a critical section, a service for that has has been designed.

A crossing simulator has been implemented so the collaborative driving management system can be evaluated. It can handle multiple rotations of 3-way intersections and roundabouts while supporting randomized directions of travel, starting distances, and turns.

For the simulation of the system, it has been rebuilt so that there can be a dynamic number of trucks that each have their own node. The code for visualizing the trucks has been modularized to follow the Model View Controller (MVC) pattern, where the logic behind the truck is in the model. The visualization layer communicates with RViz to make them appear on the screen, thus being the view, and the automatic control gives the truck steering signals and acts as a controller.

The road system was broken out of the visualization and put into its own map data server so that it can easily be replaced.

Chapter 2

Theoretical Background

This chapter introduces the theory behind the path planning algorithm and collaborative driving management, both of which are important components of the system.

2.1 Multi-Agent Path Finding

Path finding is a type of problem focusing on finding a route between two points on a network map. Given a system and an agent in that system, single-agent path finding problems task themselves with finding an optimal route for that agent from its start position to an end position.

If there are multiple agents within a system, all having start and end positions, the problem becomes a *multi-agent path finding* problem (MAPF) and aims to find an optimal route for each agent. However, a major constraint of MAPF systems is that a position can only be occupied by at most one agent at a given time. Solving a MAPF problem requires the solution to account for this and provide a non-conflicting path for each agent, where a path is a sequence of actions that the agent should take to reach its goal.

MAPF algorithms are divided into two classes, *optimal* and *sub-optimal* [10]. Optimal algorithms always yield the best solution where all agents reach their destination with an optimal solution, under the constraint that no collisions occur [22]. An optimal solution varies depending on the scenario and what is being optimized; time, distance, fuel usage, etc. Optimal algorithms are NP-hard because

their state space grows exponentially with the number of agents. Sub-optimal algorithms, on the other hand, aim to quickly find a path for all agents, without guaranteeing optimality.

With multiple agents in the system, the shortest paths may not always be the fastest, for example due to traffic congestion. To solve a MAPF problem, actions that are not usually present in single-agent path finding also have to be considered, such as waiting at critical sections, taking a detour, or moving out of the way for another agent. If aiming to find the lowest travel time, using a sub-optimal solver could lead to increased travel time due to detours were simply waiting might have been a better option.

Optimal algorithms are used when there are few agents and a minimal-cost solution is desired [10]. Variants of the A* algorithm can be used to solve multi-agent path finding. A* has an exponential state-space, i.e. its memory usage grows exponentially with the number of agents present [10], [23]. To utilize a path-planning algorithm for a system with multiple agents, it is important to consider that time elapses continuously, as opposed to many classical path-planning algorithms which model time using discrete time steps.

For the scope of this project, a suitable algorithm should be able to handle continuous data rather than discrete. For this reason, a continuous variant of Conflict Based Search (CBS) known as Continuous Conflict Based Search (CCBS) was chosen [10], [12].

2.1.1 Continuous Conflict Based Search

In contrast to the exponential state space some MAPF algorithms such as A* suffer from, in single-agent path finding, there is only one agent and the state space is linear to the size of the graph. CBS solves MAPF by applying constraints to multiple single-agent path finding problems. A constraint in CCBS is a limitation put on the single-agent path finding problem to resolve conflicts, of the form "agent A shall not be in position X during a time interval $t1 < t2$".

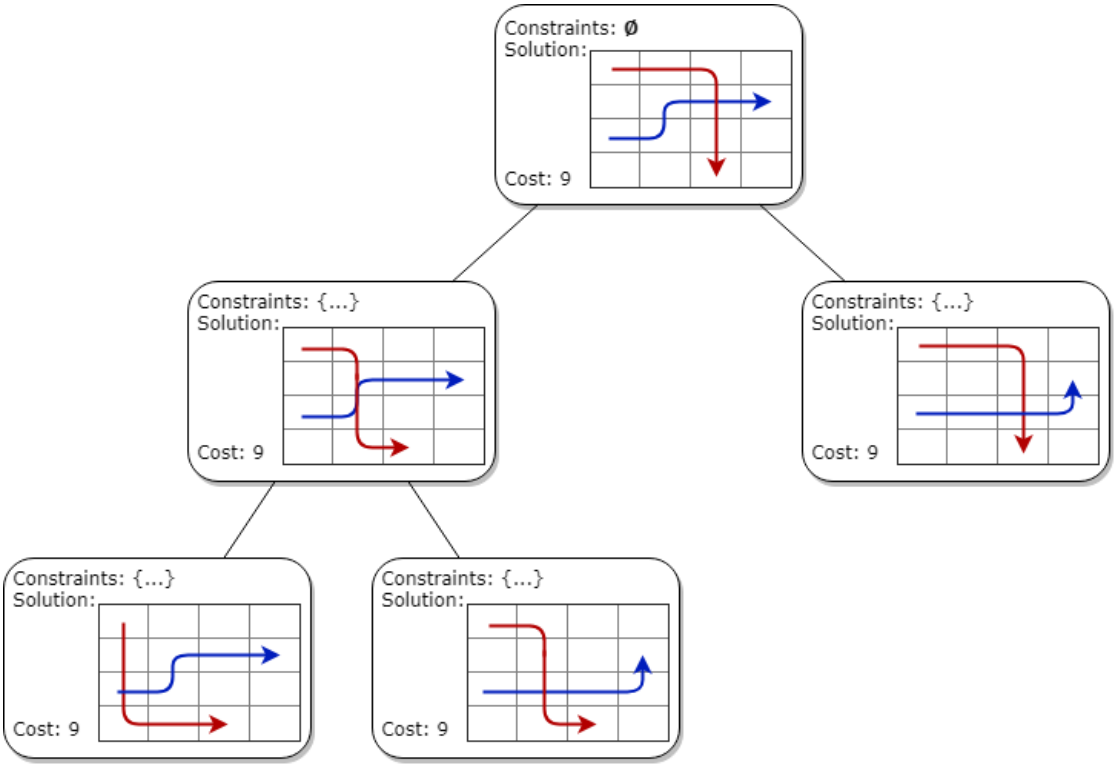


Fig. 2.1: Example of a CBS constraint tree with two agents trying to reach their respective goal. Each node contains a set of constraints, a solution, and the solutions cost.

CBS operates on two levels; high- and low-level. At the high level, CBS manages a binary tree called the *constraint tree* shown in Fig. 2.1 where each node of the tree contains a set of constraints that apply to this node, a solution consisting of agent-specific paths, and the total cost of the solution in the node. The root node has no constraints and each child node inherits all constraints from its parents and adds one new constraint for a single agent.

With the list of constraints in a node, a low-level search is invoked which calculates one shortest path for each agent that is consistent with all the constraints. These paths are validated against each other, and if no conflicts are found, the paths are returned as a solution to the problem. A solution for a conflict is valid only if at most one agent occupies any given node at any time. If a conflict between two paths is detected, the node containing the conflict spawns two new children, each with new constraints corresponding to the agents involved in the conflict. Agents have a given geometric size that the algorithm uses to check if a conflict occurs.

The single-agent path finding algorithm that provides the low-level search is executed for every agent and the set of constraints that apply to that agent. For compatibility with the scope to have a continuous algorithm, *Safe Interval Path Planning* (SIPP) was used as the low-level search, described in detail in the following section.

2.1.2 Safe Interval Path Planning

Based on A*, SIPP is an optimal path planning algorithm that considers time intervals instead of discrete-time steps like A* when looking for collisions with dynamic obstacles [11]. SIPP considers two types of intervals; safe intervals and collision intervals. A safe interval is a period where an agent can either move to or wait at a node without collisions, while a collision interval is the inverse. A collision interval is defined by a continuous period that is preceded and followed by safe intervals.

SIPP is used by the CBS algorithm to solve the single-agent path planning problem given a set of constraints that directly correspond to boundaries of collision intervals. SIPP considers all neighboring nodes and calculates the earliest time it can reach them without a collision occurring. This iterative process is repeated until the desired node is reached, whereupon the path with the lowest cost is returned. The amount of collision intervals and safe intervals is often significantly lower than the number of discrete time steps, therefore requiring fewer states to be visited, thus making the algorithm much faster.

2.2 Collaborative Driving Management

Collaborative Driving Management (CDM) plays a key role in guaranteeing safety in traffic. Areas of the road network where vehicles traveling in different directions intersect have the highest risk for collisions to occur [24]. Such areas can be intersections, roundabouts, or narrow roads. Figure 2.2 demonstrates a dangerous situation in 4-way intersection.

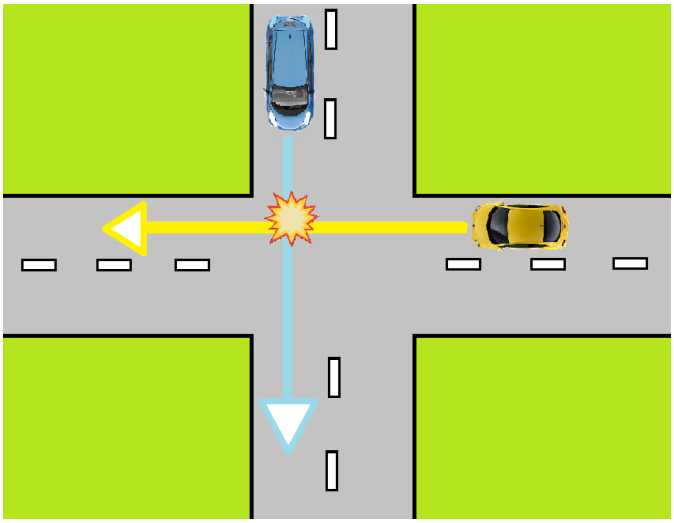
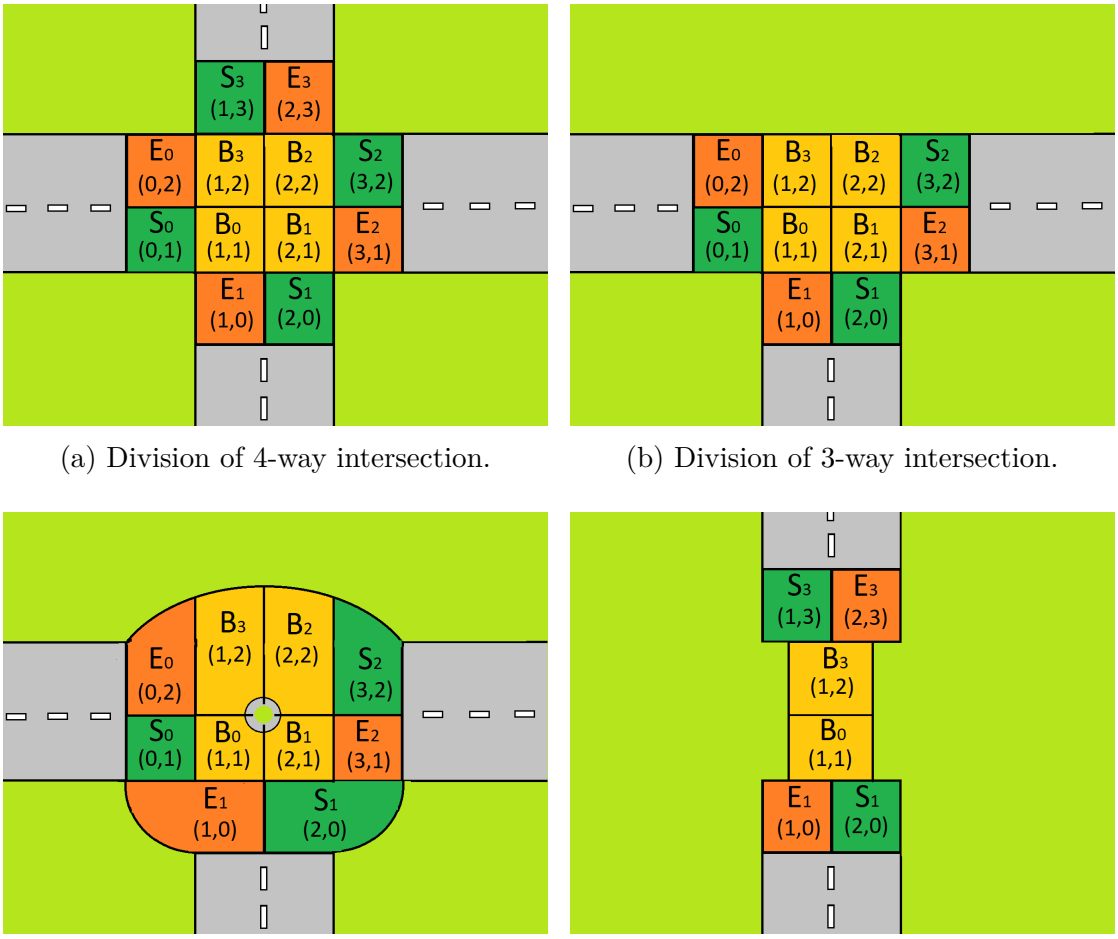


Fig. 2.2: Potential risk in 4-way intersection: the vehicles might collide while traversing the intersection.

In order to minimize the risk of collisions in such sections, the traffic in these sections needs to be managed properly. This is the role of the CDM system, which aims to minimize these risks. The CDM system extensively uses the concept of Virtual Traffic Lights (VTL) in order to coordinate movement inside critical sections. The system automatically manages and controls the movement of the vehicle in such sections, by calculating schedules for every vehicle in advance and allowing it to traverse the section at the right time in order to prevent conflicts. CDM creates a holistic system in a given enclosed traffic region consisting of different types of critical sections by implementing the VTL inside every critical section taking into account the distinctive characteristics of every section. This helps to achieve safety in the given region.

2.2.1 Defining Critical Sections

To manage the critical sections and vehicles traveling through them, each critical section is divided into several square subsections [25]. Each of these squares has a pair of coordinates and represents an area that a vehicle can occupy at a specific time. Fig. 2.3 demonstrates how the considered critical sections are divided.



(a) Division of 4-way intersection.

(b) Division of 3-way intersection.

(c) Division of a 3-way roundabout.

(d) Division of a narrow road.

Fig. 2.3: Divisions of critical sections into subsections to model their layout. Entries (S) and exits (E) to the critical section are defined, and the critical area is divided into blocks (B).

Fig. 2.4: 3-Way intersection divided into small squares in the lab room

All legal paths that a vehicle can take through an intersection can be represented by the set of squares that are occupied during the trip through the intersection. Considering Fig. 2.4, a vehicle starting at position S1 and intending to travel to E3 will occupy B1, B2, and B4. For each type of intersection, all possible path inside this intersection has been defined.

2.2.2 Avoiding Conflicts

Conflicts can occur when several vehicles are following the same subset of activated squares simultaneously. See Fig. 2.5 for an example.

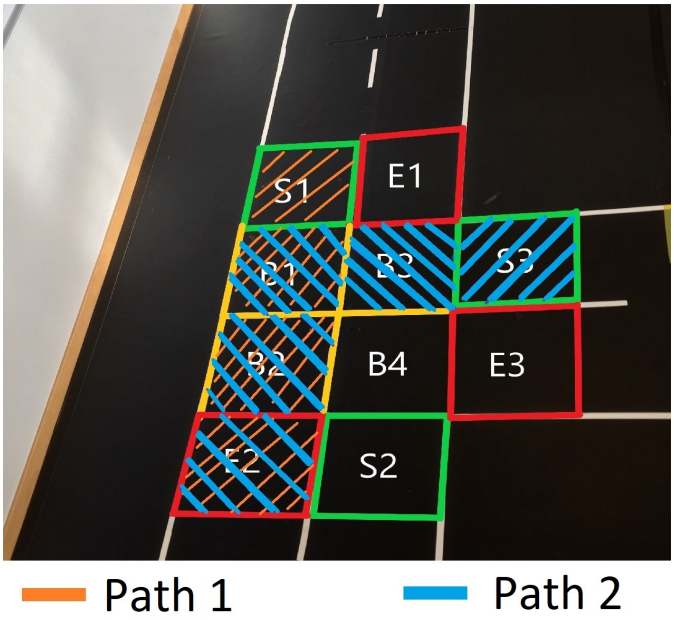
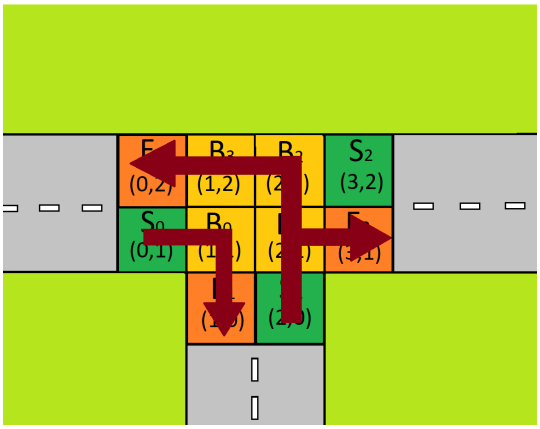
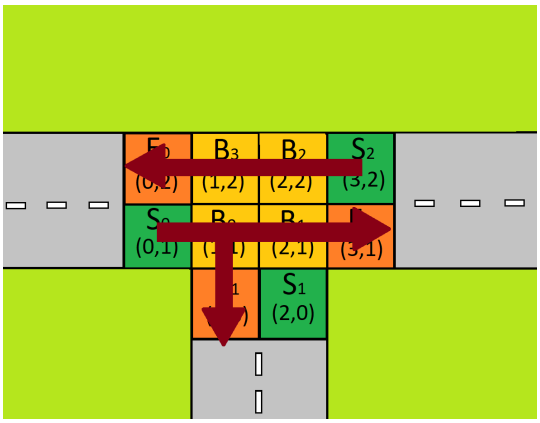


Fig. 2.5: Occupied squares for two intersecting paths. The activation of both paths at the same time will lead to a conflict in one of the squares B1, B2, or E2.

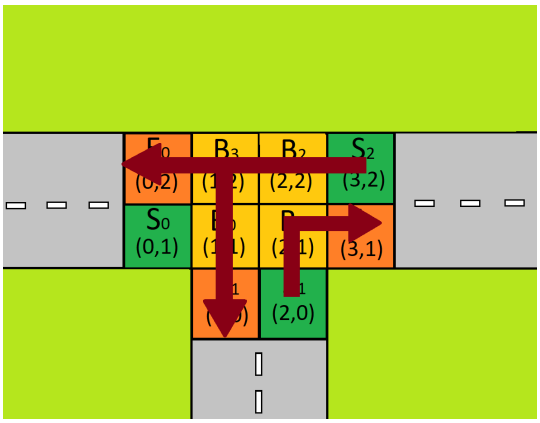
To prevent conflicts, any subset of the activated squares inside the path may only be occupied by at most one vehicle at the same time. Considering the set of all possible paths through the intersection, the algorithm presented by Strid *et al.* was used to pre-calculate so-called *non-conflicting sets* [7]. Each non-conflicting set consists of all paths that do not conflict, i.e. active at the same time while guaranteeing no collisions, and the sets could therefore be described as being closed under non-intersection. This means that the non-conflicting sets must differ for each type of intersection. For example, all non-conflicting sets for a 3-way intersection are shown in Fig. 2.6.



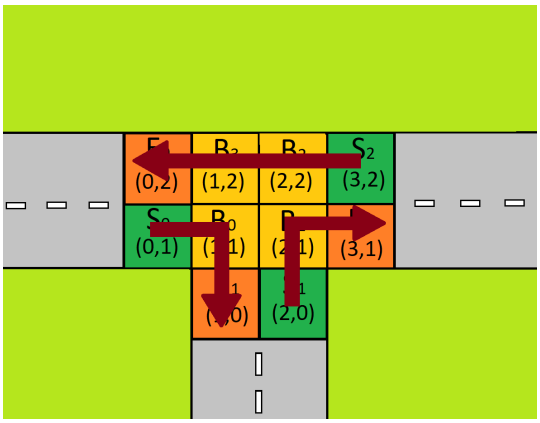
(a) Paths: South → East/West, West → South.



(b) Paths: East → West, West → South/East



(c) Paths: East → West/South, South → East.



(d) Paths: East → West, West → South, South → East.

Fig. 2.6: Four different non-conflicting sets in a 3-way intersection.

When a vehicle approaches a critical section, the CDM makes the following decisions to coordinate movement and bring the vehicle safely through the region: the type of critical section, the non-conflicting sets for this critical section, and how many vehicles are involved in the critical section at the moment.

When these things are decided, the system creates a schedule consisting of a subset of the paths from the given *non-conflicting set* to be activated a d time to be activated for each path from the subset. Afterward, the chosen paths activate in turn and order according to their decided time and the vehicles follow these paths, which allow them to cross the intersection safely.

Chapter 3

System Architecture

This section describes the overall architecture, what components it consists of, how they interact, and work.

3.1 Overview and Guiding Concepts

When designing a system to solve such a complex task, a good design is critical for success. The proposed system consists of components, separating distinct concerns and features for modularity and in accordance with good software engineering principles. An overview of the system design in terms of components and their interactions is shown in Fig. 3.1. The components of the system are the vehicles, a map data server, the path planner, and Collaborative Driving Management (CDM) for critical sections. These components are described in more detail in the upcoming sections. The modularity of the system allows for simpler testing and verification because it makes it possible to treat components individually rather than in interaction with each other. Further, the modular design makes adapting the behavior of the system to different requirements easier, such as updating the road network graph or introducing different types of vehicles.

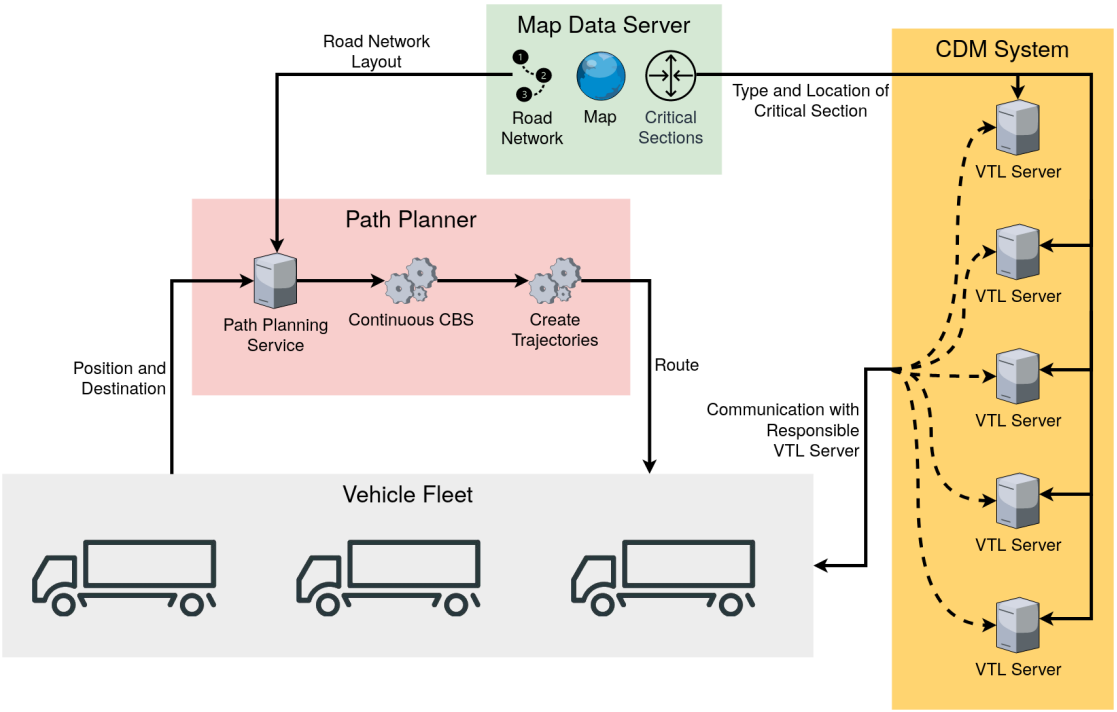


Fig. 3.1: Overview of the system architecture. The four main components of the system are the vehicle fleet, the map data server, the path planner, and the Collaborative Driving Management (CDM) system which contains several Virtual Traffic Light (VTL) servers. The communication and information flow between these components is also shown.

During startup, the map data server provides the path planner component with information about the road network and provides the collaborative driving management system with information about the types and locations of critical sections.

After the system has started, vehicles submit requests to the path planning service and, once they have received their respective route, start their journeys. The vehicles are fully autonomous but lack knowledge of the positions and intentions of other vehicles, which drastically reduces their complexity. To guarantee safety at critical sections where collisions between vehicles may occur, the CDM system is used. When approaching a critical section such as an intersection, the vehicle will contact the virtual CDM server responsible for that area. The server will then arbitrate the order in which any vehicles will traverse the critical section, thereby eliminating the risk of collisions. The individual components and their behavior is described in more detail in the upcoming sections.

3.2 Map Data

The map data server is the authoritative component regarding information about the static environment. At startup, it loads all assets related to the environment, including an image of the area from a top-down perspective, a directed multigraph of the road network, and the types and positions of critical sections. This information about the system can then be queried by other components of the system. All assets for an environment are stored together in a directory, making it simple to switch out the map.

3.3 Vehicles

The vehicle agents model the behavior of individual vehicles in traffic. Each vehicle instance stores its location and orientation, the combination of which is commonly referred to as *pose* in robotics, as well as velocity and destination. Vehicles submit their pose and requested destination to the path planning service at startup, and in response receive a route that they will then follow. Driving and control is fully autonomous, and is based on continuously minimizing the error in position and bearing between the current location and the next point on the route. When the vehicle approaches a critical section, it will connect to the responsible CDM server for that region to ensure safety during the traversal of the section.

3.4 Path Planning for Multiple Agents

At startup, the path planning server requests a road network graph from the map data server and receives the number of vehicles present in the system. Afterward, the server waits for all vehicles to submit their requests, consisting of their current position and destination. Once requests from all vehicles have been received, the path planner, based on CCBS as described in Section 2.1.1, is invoked. Once a solution is found the server responds to the back the calculated paths to the trucks. This process is illustrated in Fig. 3.2.

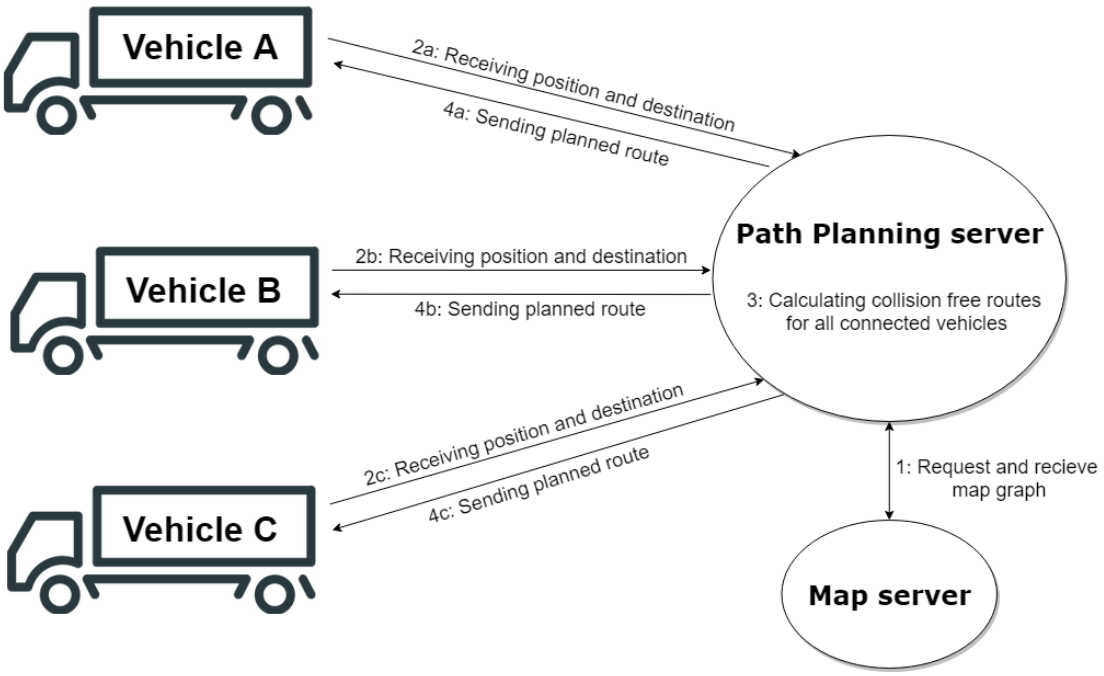


Fig. 3.2: Layout and steps of the path planning system.

The solution calculated by the path planner is a set of paths through the road network graph, called the *reference paths*, with one such path for each vehicle. However, the reference path cannot be followed by a vehicle because it would require impossible maneuvers such as 90° turns on the spot. Each reference path is therefore transformed into a *trajectory* for the vehicle to follow, taking into consideration their capabilities and physical limitations.

Once the vehicles have started traveling, they will follow their given trajectory until they have reached their destination. Due to the continuous nature of the CCBS implementation, timing amongst vehicles is critical during execution, since any deviations would obsolete all predictions and prevention of collisions made by the planner. However, should a vehicle become delayed, all vehicles will continue according to their respective plans, while safety will still be guaranteed by the CDM system, which acts as a fallback safeguard in this situation.

3.5 Collaborative Driving Management (CDM)

The CDM component is built as a server-client structure, where each critical section has a server allocated to it (see the figure 3.3) and every vehicle in the system has a client for these servers. When starting a server, the type of critical section is specified along with the directions from which traffic should be able to enter and exit. The system can manage the following types of critical sections: multi-road intersections, which include 3-way and 4-way intersections, as well as 3-way roundabouts and narrow roads.

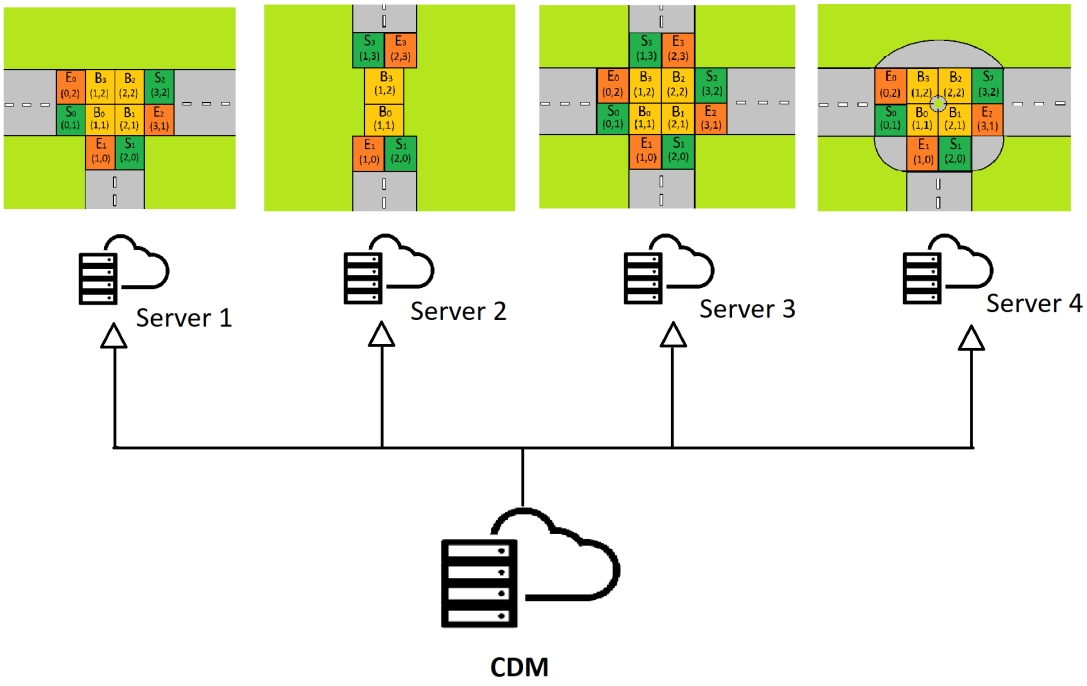


Fig. 3.3: Every intersection is handled by a dedicated VTL server.

All types of critical sections are designed as a set of separate modules, which are chosen dynamically at startup based on the type of critical section the server should handle. This allows the system to apply different characteristics to different critical sections and leaves the potential to add more diverse functionality in the future, such as more types and variations of intersections. The possible paths in the intersection are calculated dynamically along with the non-conflicting sets.

To allow for vehicles to connect to different servers a connection system was put in place. Inspired by the work of Andréasson *et al.*, a critical section zone and connection zones for each critical section was defined [8]. The critical section zone

outlines the whole critical section while the connection zones outline a rectangle along the entrance roads leading up to the critical area. While a vehicle is traveling, it frequently checks whether it is located within any of the connection zones. As soon as the client identifies that it is inside a connection zone for a critical section, it connects to the responsible VTL server and starts communication with it. Fig. 3.4 is a simplified model visualizing this communication.

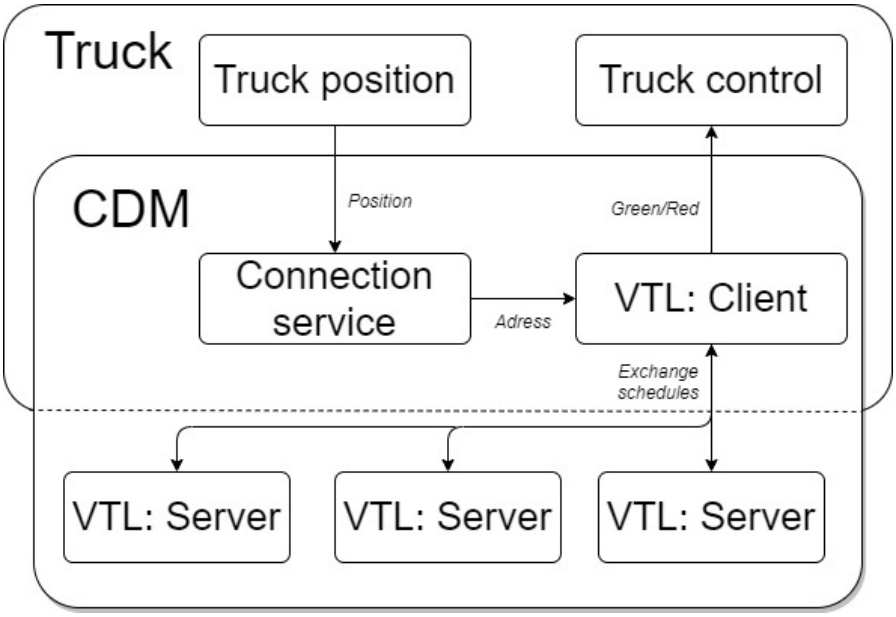


Fig. 3.4: Server-Client communication. When the connection service notices that the vehicle is approaching an intersection, it sends the server address to the vehicle, which connects to that server and enables communication with it. When the schedule tells the vehicle that it is allowed to go, the client sends a green signal to truck control.

As soon as the communication between the client and the intersection server is enabled, the client notifies the server of its path through the intersection, i.e. what direction it intends to take. Then, the server sends the schedule to the vehicle. This process is illustrated in Fig. 3.5.

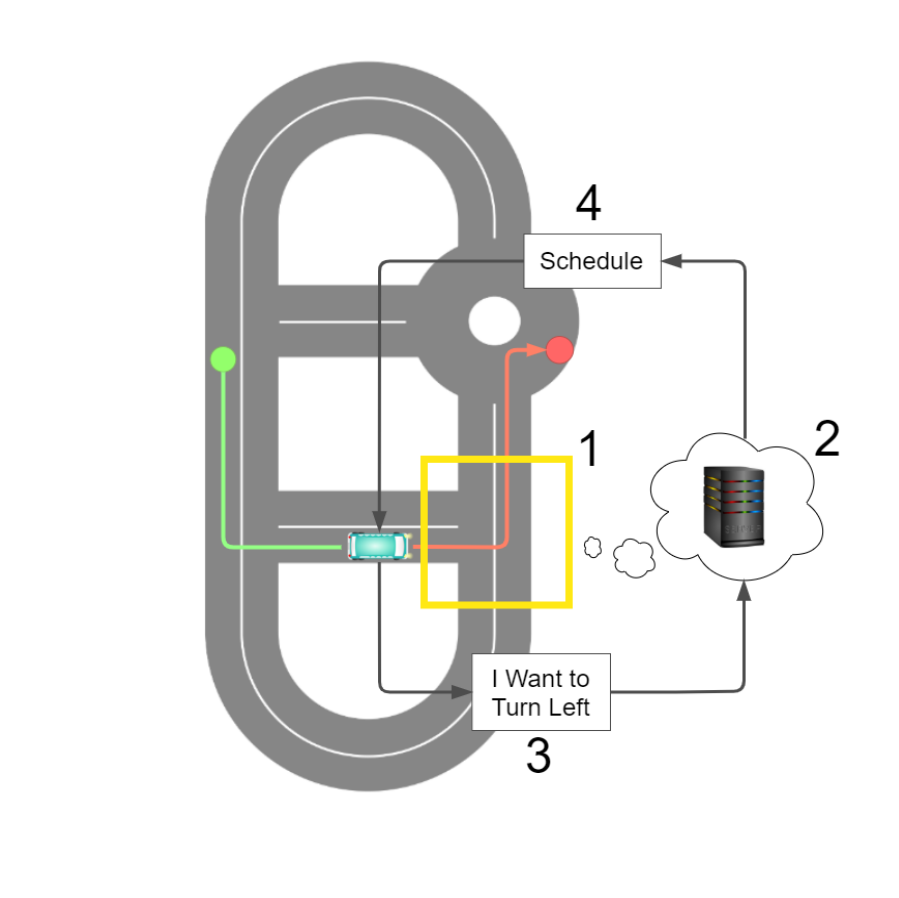


Fig. 3.5: Client-Server communication in the intersection. 1: Intersection area; 2: Local server responsible for the given intersection; 3: A client approaching the intersection sends its planned path to the server; 4: Server sends schedule to the client.

The main purpose of the server is to continuously calculate and send its schedule to the clients. When the server has received the path of the vehicles inside of the intersection, it sends this schedule. The schedule consists of three phases: now, next and tentative. Each phase in turn consists of a non-conflicting set, a time for how long this set will be active and agreement flag which is true if all clients have accepted the server scheme-proposal. When the phase moves on to next from tentative the flag cannot be changed anymore.

After the server calculated the schedule, it sends the schedule to every vehicle. The server also continuously checks if the clients have the same tentative phase, i.e. the

same set of paths and duration time. If they do, the tentative agreement flag is set to true. In case the agreement flag is not set to true before the tentative phase becomes next-phase, indicating that not all vehicles share the same schedule, the phase is disregarded.

On the client-side, after the vehicle has sent its path it waits for the schedule from the server. When the client receives the schedule it sends back the schedule to notify the server that the schedule has arrived. If the client approves the schedule, it sends back the schedule to the server, so the server knows that the constructed schedule has been accepted by the client. Afterward, the client uses this scheme in its movement through the intersection until it gets a new.

Once the vehicle has left the intersection, it sends the last message to notify the server that it has exited the critical section. The server removes the agent from its list of agents in the intersection and the communication stops.

The system implements VTL in every critical section, by creating a VTL server in every such area. The client is responsible for its path and movement. Every time the client detects its location is within the critical section, it starts communication with the responsible server. The client then performs the movement through the section according to the schedule received from the server.

Chapter 4

Method

This chapter describes the workflow used by the team during implementation of the prototype system and the methods used to collect quantitative data about the performance of the system.

4.1 Software Engineering

During the project, an agile workflow was used to create an iterative process within the project. The group split into three sub-teams having different areas of responsibility during the project: Collaborative Driving Management, Multi-Agent Path Finding, and Demonstration. At the beginning of each working week, the group had a planning meeting to discuss progress, tasks to complete during the upcoming week, and personal well-being. This process meant that all group members were up to date with the overall progress of the project and how the group was feeling, so changes could be made to make ensure everyone was as productive as possible while maintaining healthy work procedures.

Version control was managed with Git on the BitBucket platform, and since the system has a modular design, these were developed in separate repositories. The GitFlow workflow, shown in Fig. 4.1, was used to ensure that each team always had a master branch with a working code. This also allowed the team members to work on different branches, consequently encouraging designing features as vertical slices where each feature is a fully functional slice of the program.

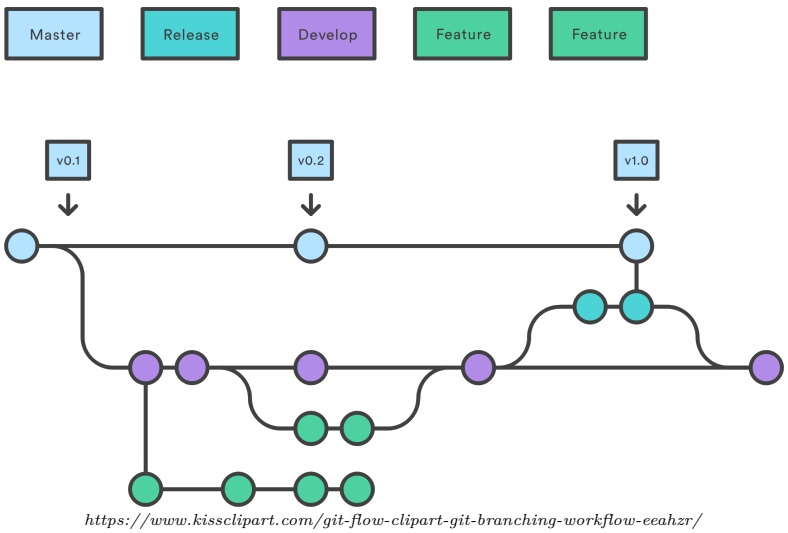


Fig. 4.1: Visualization of the GitFlow workflow. Master is only pushed to with functional code from the development branch, all new features are branched off while being developed then merged into the development branch.

To keep track of tasks and progress, each team used a Trello board hosting their project backlog consisting of user-stories and tasks. This helped the teams in the early stages to break down the system into smaller components, analyze and plan the architecture. Having three different backlogs gave an overview of the planning and progress of each part, which helped with the system integration of the modules.

Following the recommendation of the Public Health Agency of Sweden concerning the outbreak of COVID-19, the group adapted the workflow so that all work was done remotely. This was a new challenge for the whole group, and daily meetups in the morning were introduced to uphold daily communication and progress. During the meetings, everyone shared what they were working on that day, and if any problems had emerged. To encourage achieving weekly goals a daily workshop was scheduled as well. The weekly meetings continued to take place, but were also carried out remotely.

4.2 Measuring System Performance

To assess the performance of the implemented system, quantitative data will be collected.

4.2.1 Benchmarking CCBS

For a system handling automated vehicles a path planner must find solutions quickly and reliably. To measure how well the CCBS perform within these criteria, testing was conducted on a network map representing a vehicle road map seen in Fig. 4.2. This map was taken from a previous bachelor project group used to simulate the physical road-map in the lab room. The map was modified and consists of 412 nodes, 100 new nodes were added on the original to be used as exclusive start and destination positions shown in red. More start/destination pockets could be added, and as such more agents, but due to the manual labor needed together with time constraints, this was not done. Three different map sizes based on the one shown in Fig. 4.2 were used to test the system, a small, medium and large version with an respective size of $88x41$, $880x410$ and $8880x4110$ area units.

The agent size is represented as a circle, so to make things simple the diameter was set to one unit of length thus each agent occupy π unit of area. For testing the scalability of the system tests were done with up to 100 agents on each map size. Making sure the results were rigorous, each test case ran 25 times where every agents start and destination position were randomized.

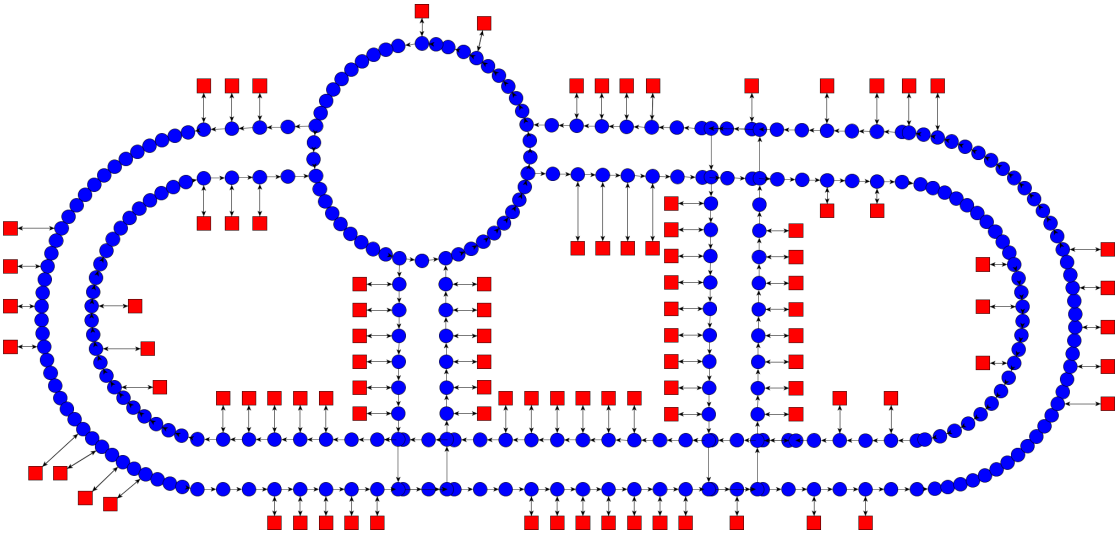


Fig. 4.2: Overview of the Node network map used for testing the path planner. The red squares represent the start and destination positions.

4.2.2 Benchmarking CDM

The movement of vehicles in critical sections, controlled by the CDM, depends on frequent updates of the VTL servers. The frequency with which these state updates occur, the number of cars involved in traffic, as well as type of intersection, have an impact on the number of cars able to cross the section, which directly corresponds to traffic flow in the area. In order to evaluate how the performance of the CDM system is affected by these parameters, two experiments were conducted.

The critical sections that participated in the test were 3-way intersection, 3-way roundabout and 4-way intersection. The reason behind was that at that time, when the experiment was executed, these were the sections that were fully implemented. The time for the activity of each set, T_set, was set to a constant value of 0.5 s, since this becomes the server's reaction time this interval was considered a good compromise between diminishing return and optimality.

To evaluate the scalability of the system, different numbers of active vehicles were tested. It was decided that the number of cars should be 1, 10, 20, 30, 40 and 50 for each experiment. The reason behind this choice was that this range of numbers helps to see the relation between the number of cars and the server's efficiency. In the mentioned intersections, 50 vehicles can be considered as an upper bound to be involved at the same time, especially with respect to how the CCBS system plan the vehicles' paths.

Then, the update frequency, T_period, of the VTL servers was changed, and each frequency was tested with the previously mentioned values for traffic. The values for server's period were chosen as following: 0.1 s, 0.3 s, 0.5 s The reason behind this choice was to have a variety of numbers and at the same time to not have values on frequency bigger than on set activity, which would render it impossible for the system to ever reach an agreement due to its design. This choice of values could help to see the relationship between different frequencies and their impact on the client's ability to accept the agreement flag. This affects how often clients are able to positively accept phases in critical sections and agree upon a schedule.

Time for the test execution was 60 sec, which is a realistic amount of time needed for up to 50 vehicles to pass the intersection. The number of successful traversals was then logged for each of the tests done.

Chapter 5

Evaluation Environment

In order to evaluate the behavior and performance of the proposed system, a prototype was implemented. Since testing in a physical environment with vehicles or models was not feasible, a simulative approach was used for evaluation.

5.1 Software

The following software and tools are used to implement the prototype system. The choice of tools is generally based on their suitability for the purpose and the experiences and work of previous groups working on related projects.

5.1.1 Robot Operating System

Robot Operating System, or ROS, is an open-source, meta-operating system commonly used to implement robotics systems [26]. ROS systems are composed out of communicating processes called *nodes*, which makes them highly suited for implementing a distributed modular system. ROS provides a layer of communication between all nodes to communicate according to a publisher/subscriber pattern, where nodes can publish and subscribe to different topics. By subscribing and publishing to topics, the individual nodes form a distributed system called the *ROS graph*, and can then cooperate to accomplish tasks. ROS also offers communication via *services*, which function on a one-off basis; a client submits a request to a service, and the service provider responds. Topics are suitable for streaming data continuously, such as a truck sharing its state, while services are used for remote procedure calls that might also involve computation of data, such as a path

planner.

5.1.2 RViz

RViz is a graphical tool for ROS that is used for 3D visualization of data [27], and forms the backbone of the simulators implemented in this project. It can visualize many different types of data such as primitive shapes, so-called *markers*, as well as paths, points, and many others. By positioning the camera along the z-axis, a top-down perspective of the environment is obtained, where vehicles represented by markers can be observed as they move.

RViz allows using individual coordinate systems for each element that it is drawing by using the geometric transformation system `tf2` [28]. Using `tf2` allows placing each visual component such as the map, truck cabs, and truck trailers within an independent coordinate system. These coordinate systems are then linked into a *transform tree*, which tells RViz how to transform each coordinate frame relative to the `world` coordinate frame which always remains stationary. A small example of the transform tree used in the project is shown in Fig. 5.1.

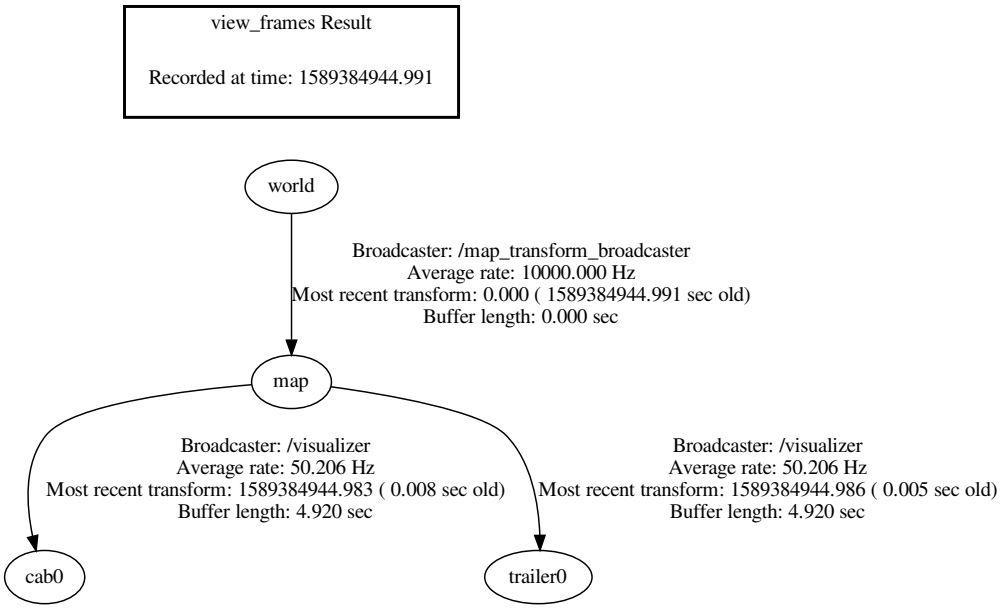


Fig. 5.1: Transform tree created by `tf2` showing how the `cab0` and `trailer0` frames related to the `map` frame, which in turn relates to the constant `world` frame. Thus, a truck can be projected onto the world by applying the transformations along the path to the root node of the tree. More `cab#` and `trailer#` nodes are added under `map` for subsequent trucks.

Using separate coordinate frames for each component makes it very straightforward to work with coordinates relative to that component, such as finding points straight ahead of trucks. Through the transform tree, these coordinates can then be transformed into other coordinate frames. This is used extensively during the visualization of vehicle movement, where each vehicle always remains at the origin of its frame, while the frame is translated and rotated relative to the map. Using separate frames also introduces the ability to attach the camera to these frames instead, which could yield interesting perspectives such as following individual trucks or first-person views from truck cabs.

5.1.3 Programming Languages

ROS allows using different languages to implement nodes, with the most common ones being Python and C++, because those are often preferred in robotics development. The prototype system is implemented entirely in Python, which was chosen due to its shallower learning curve as opposed to C++, and because most of the previously existing code was written in Python as well. The system mainly uses Python 3 due to the extensive index of available third-party libraries and tools, but all visualization code had to be implemented in Python 2 to work around missing support in the `tf2` library. This will be remedied in a future version of ROS, but the team valued the stability of a released version of ROS over using a developmental version with newer, potentially buggy, features.

5.2 Collaborative Driving Management Simulator

Simulation has been used continuously during the whole development stage of the CMD. Since traffic coordination in critical sections is the most important part of the drivers' safety, a simulator was built. In the simulator safety and efficiency of the system could be tested by constructing and visualizing different types of critical sections. The simulator was also used to find abnormal and unexpected behavior alongside testing how the system handles agents when they do not behave as expected.

Fig. 5.2 shows a screenshot of the simulator simulating a 3-way intersection. Each colored line is the intended path of the vehicle with the corresponding color. The green arrows correspond to the active non-conflicting set and show which paths are currently active. In this case, the cars are allowed to drive from north to east or west and from east to north.

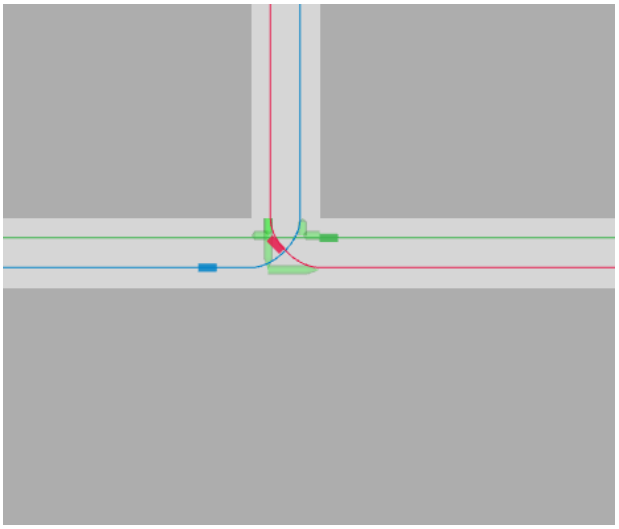


Fig. 5.2: Screenshot of CDM simulator.

5.3 System Implementation

In order to assess the merits of the proposed system design, a prototype implementation was made, including all components as described in Chapter 3 as well as certain additional components for visualization.

5.3.1 Visualization

The visualization has been implemented next to the prototype system in order to provide a view of the systems current state. Two components comprise this part of the system – a visualizer that is responsible for gathering information about the state of the system, and RViz which is used as the graphical front end for the system.

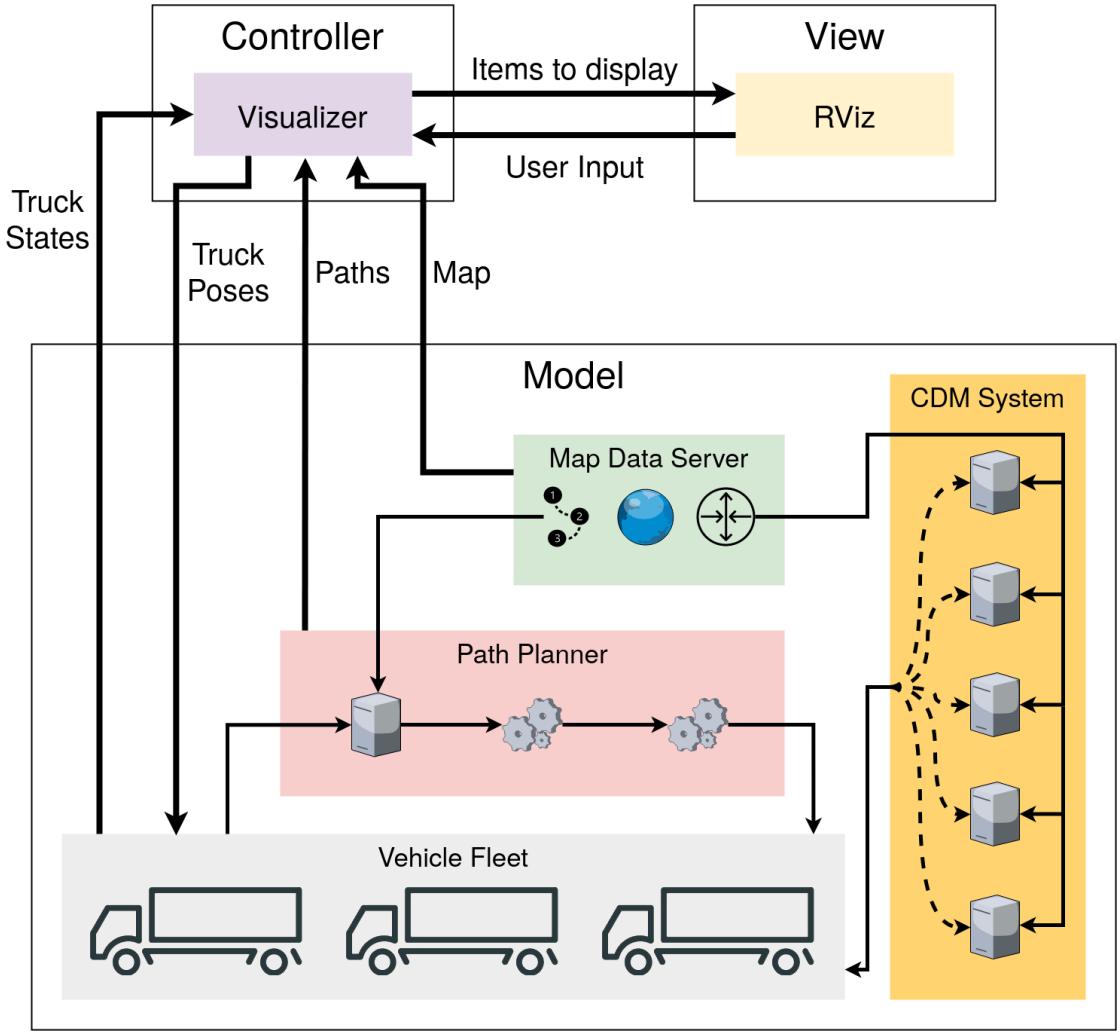


Fig. 5.3: System visualization architecture using the MVC pattern. The model is equivalent to Fig. 3.1

As illustrated in Fig. 5.3, the full architecture is based on the *Model-View-Controller* (MVC) design pattern. The complete prototype implementation contains all components described in Chapter 3 and forms the Model component of MVC. It is responsible for the data and logic of the system, such as modeling vehicle dynamics or planning paths. Along with it, a visualizer node, a transform broadcaster node, and an RViz node are also present in the system. The visualizer node forms the Controller component of the MVC structure by subscribing to a number of topics in the model where data about the state of the system is transferred, and upon receiving a message on these topics, publishes it to RViz

for display. The visualizer also receives user input from the View, such as start and destination positions for trucks, and forwards them to the model on dedicated ROS topics. RViz itself forms the View component of the system, as it contains no logic or data, and is only capable of displaying data sent by the Controller and receiving user input.

The visualizer node subscribes to a number of topics in order to receive data about the current state of the system. This information includes all `truck_state` topics, or path planner debugging-topics such as `reference_path`.It also subscribes to topics that RViz publishes user input on, such as `initialpose` and `move_base_simple/goal`, which transfer data about starting poses and destinations for trucks, respectively. The visualizer is the only part that needed to be written in Python 2 to avoid incompatibility issues with `tf2`.

The aforementioned transform broadcaster node is a tool of the `tf2` system, which allows publishing a static transformation between two frames without developing a full ROS node. Instead, it is passed a list of the translation and rotation components of the transformation and will then publish the transformation as static, indicating to `tf2` and RViz that it never changes. This is used for the transformation between the `world` and `map` frames, which is the identity transformation as no transformation should be carried out between them.

The final new component of the visualization system is RViz, which is used as the graphical front end of the system as described in Section 5.1.2. Through the *2D Pose Estimate* and *2D Nav Goal* tools of RViz, each truck is assigned a starting position and a destination position.

5.3.2 Truck Simulation

By simulating the road system and the desired number of trucks, both the Multi-Agent Path Finding and Collaborative Driving Management system can be tested.

Every vehicle in the system is composed of three nodes: one node that simulates the movement and state of the truck, one node that handles autonomous driving of the vehicle, and one node for handling communication with the CDM system. This modular design also allows for potentially implementing several other types of vehicles in the system by adapting the movement simulation node of the vehicle. The movement simulation node publishes the current state of the truck at 50 Hz to provide other systems such as the visualizer and the CDM client with information about the vehicle. These truck state updates contain the id of the vehicle, the timestamp of transmission, the vehicles position, the bearing of the cab and

trailer, as well as the current speed of the truck. Each truck node subscribes to its individual `truck#/auto_drive` topic, where the autonomous driving node for the truck publishes driving control signals. The received control messages are then executed by the simulated truck in a separate processing thread, while the main truck thread publishes the truck state at regular intervals. Because the truck node stores the current state of the truck, it is responsible for handling setting the initial pose via RViz.

The autonomous driving node, called `auto_driver`, is idle until a destination has been set in RViz, which will be received on the `truck#/destination` topic. Upon receiving a destination, `auto_driver` submits a request to the path planner service. Once a response, consisting of a list of regularly spaced (x, y) positions along the calculated trajectory, has been received, the `auto_driver` will publish driving control messages to the `truck#/auto_drive` topic, which the corresponding truck node will then receive and execute. The underlying algorithms and logic to decide which control signals to send remains unchanged from previous years projects in this environment. This is because it was deemed unnecessary to re-implement already existing functionality, and is not part of the project scope.

The CDM client of each truck consistently checks if the current position of the truck is within the connection zone for any critical section of the map. Once this is detected, the CDM client will connect to the VTL server responsible for the upcoming critical section, which will govern the next part of the journey. The `auto_driver` will keep following the generated trajectory of the vehicle, while the VTL client communicates with the VTL server to agree upon when to execute the intersection traversal, stopping the vehicle as necessary.

5.3.3 Path Planning

The `path_planner` node hosts a service of the same name, where vehicles may request a path from their current position to a destination. The service is aware of the number of vehicles in the system, and expects requests from all vehicles before path planning will commence. This is accomplished by using a barrier in the ROS service, which blocks all service request handlers up until the last vehicles request has been received. Once the final request has been received, the path planner based on CCBS is invoked with the batch of received requests, and the resulting *reference paths* through the road network are converted into *trajectories* that the trucks can physically follow. These trajectories are then transmitted back to the corresponding trucks, and the path planner service resets its state for future requests.

5.3.4 Collaborative Driving Management

The CDM system consists of two components: the VTL client placed in each vehicle, and the VTL servers responsible for each intersection. During startup of the system, information about the type and position of all critical sections is received from the map data server, and the necessary VTL servers are started. Further, the VTL client of each vehicle will also request the information about the connection zones and addresses of each VTL server and cache it. This ensures that all presence checks for critical sections can be accomplished onboard, which reduces the necessary bandwidth and reduces reliance on cloud infrastructure. Although not explicitly part of the project scope, the reliability of the CDM system is vital for safety in the system. As such, some possibilities for making the CDM system more fault-tolerant were considered, such as letting ROS automatically restart nodes that have failed, or notifying other nodes of a potential failure in the system.

Chapter 6

Results and Evaluation

This chapter presents the results obtained from several experiments carried out on the performance-critical components of the system: Multi-Agent Path-Finding and Collaborative Driving Management.

6.1 Multi-Agent Path Planner

Using the testing methods described in Section 4.2.1, tests were aimed towards finding results of two statistics; runtime and reliability. These experiments aim to determine how the performance of the algorithm scales as the number of agents is increased relative to the size of the map.

6.1.1 Runtime

To provide an acceptable experience for all clients and to avoid idle time while waiting for a route, it is important to evaluate the runtime of the path planner. If a solution is not found within the time limit of 30 s, the test is considered to be failed. Starting on the small map CCBS had no problems finding solutions up to 20 agents within a second, shown in Fig. 6.1. Beyond 40 agents the rate of collisions between agents reach critically high values, and CCBS is not able to find solutions within the time limit.

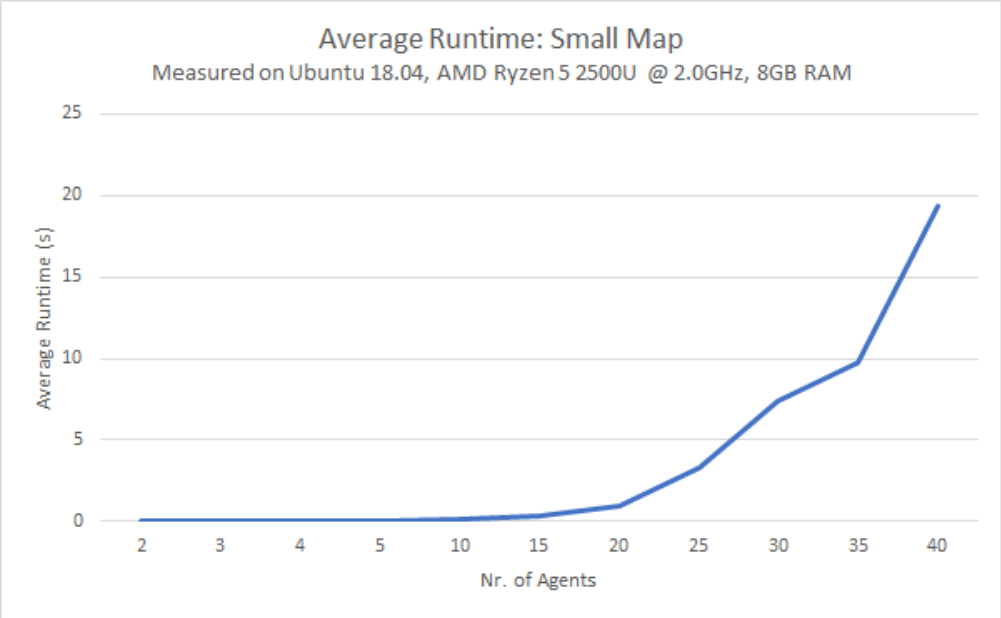


Fig. 6.1: Runtime of the CCBS on a small-sized map.

The medium map shown in Fig. 6.2 increase the hard limit of the number of agents by over a factor of two to 90 agents.

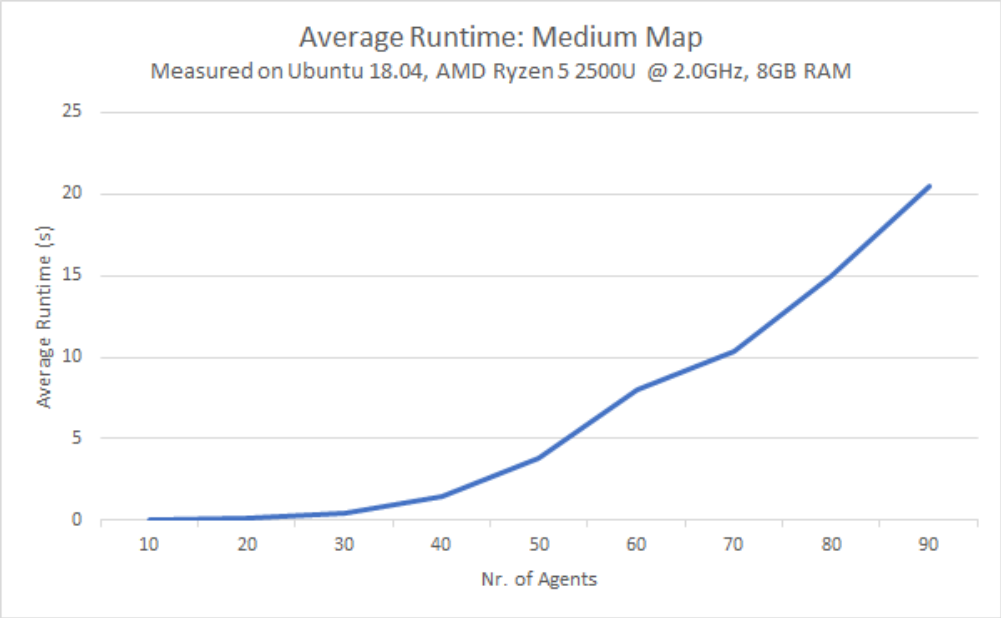


Fig. 6.2: Runtime of the CCBS on a medium-sized map.

Tests conducted on the large map are shown in Fig. 6.3. All completed well within the time limit, which is due to the much larger size of the map reducing the likelihood of collisions.

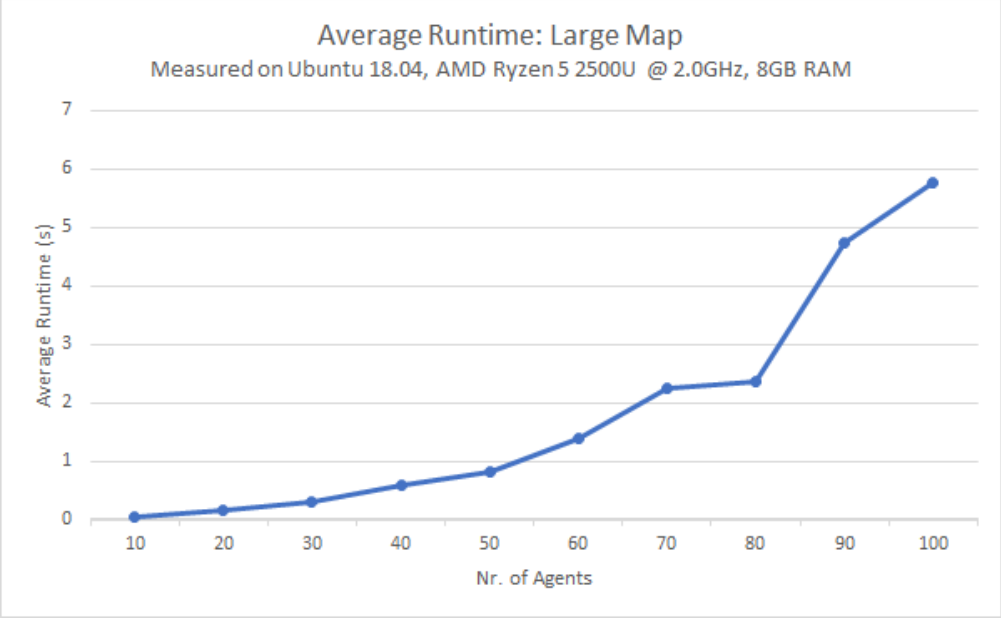


Fig. 6.3: Runtime of the CCBS on a large-sized map.

Combined, the results show an exponential increase in runtime when increasing the number of agents. Each new agent may cause new collisions that have to be worked around by creating new paths that can in turn create new collisions with other agents, and therefore, exponential growth is expected.

6.1.2 Reliability

While it is important for a path planner system to work quickly it is equally important that it always succeeds in finding solutions. A path planner that fails to find a solution can be disastrous, for example causing a system-wide stop to all connected vehicles. To measure how well CCBS performs in this regard, the ratio of successfully found solutions to total test runs was calculated.

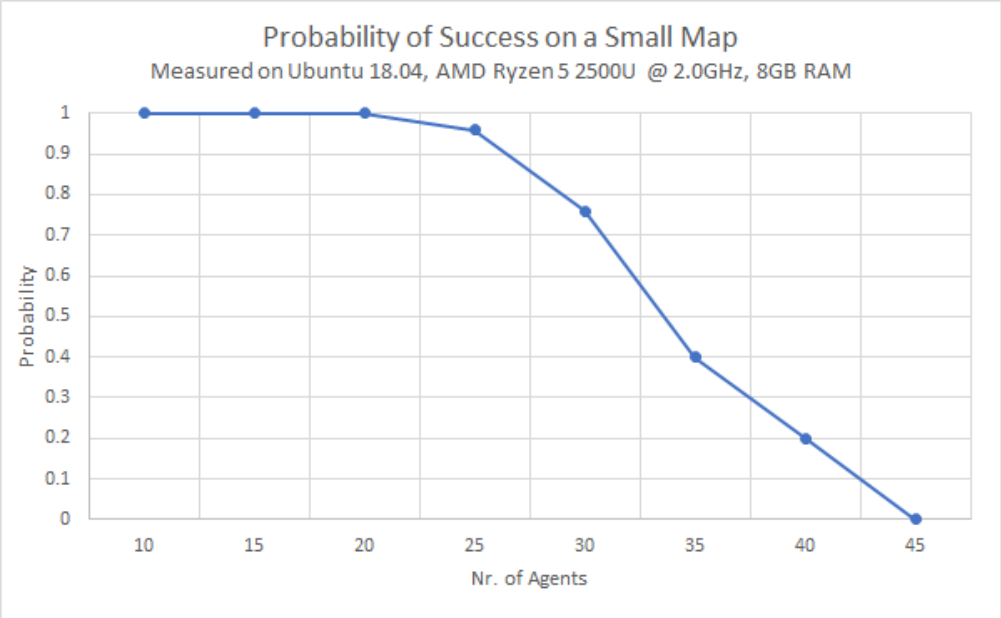


Fig. 6.4: Success rate of CCBS at finding solutions within the time limit on a small map.

The results of running CCBS on the small map are shown in Fig. 6.4. CCBS was able to handle up to 20 agents without problems. However, the rate of success falls with an increasing number of agents, and no solutions were found for more than 45 agents.

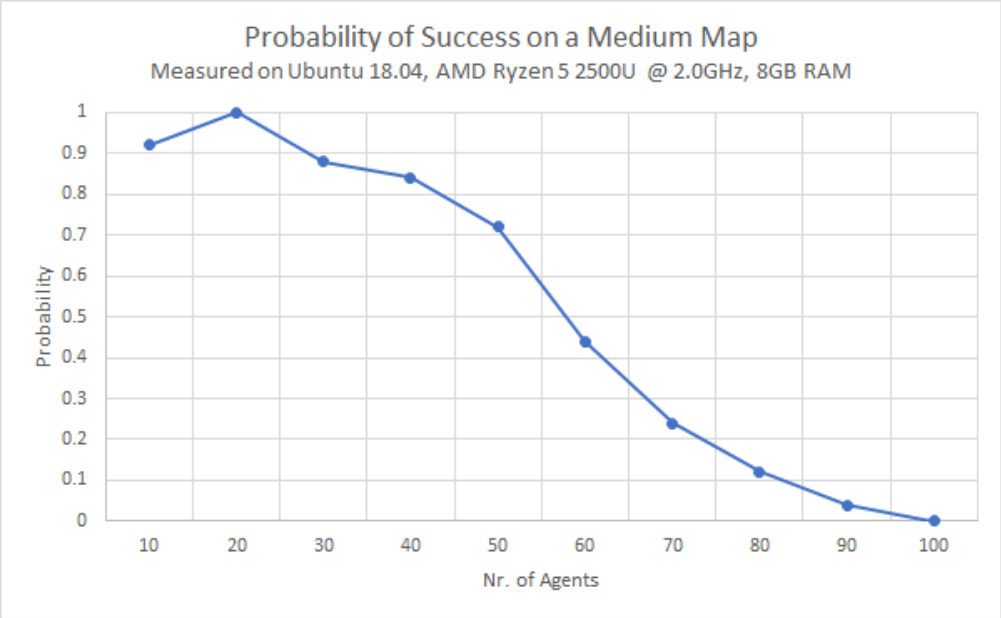


Fig. 6.5: Success rate of CCBS at finding solutions within the time limit on a medium-sized map.

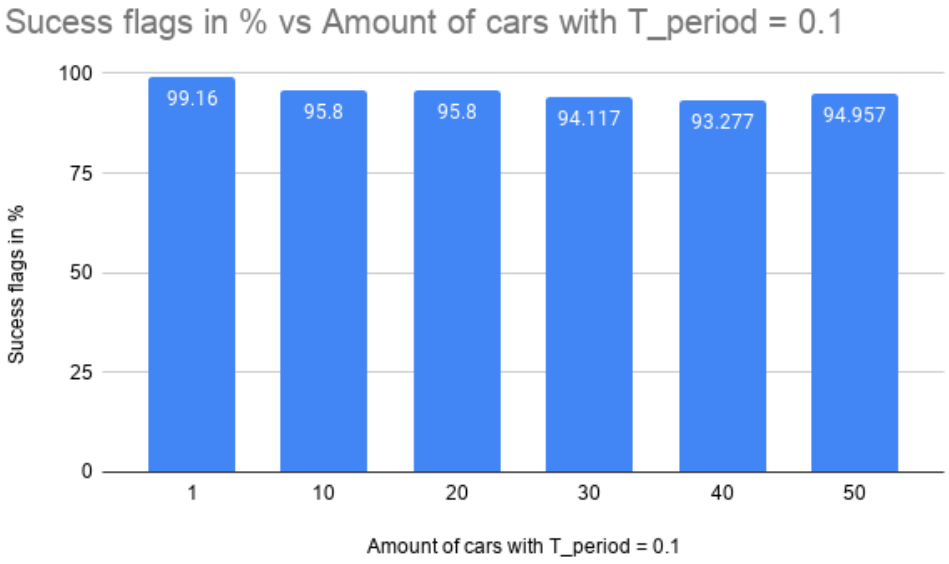
The benchmarks from the medium-sized map show some odd results when occupied with 10 agents, visible in Fig. 6.5. Otherwise CCBS starts falling off after 20 agents similarly to the small map but at a lesser rate. Every test case that ran on the large-sized map found a solution within the given time limit.

6.1.3 Observations

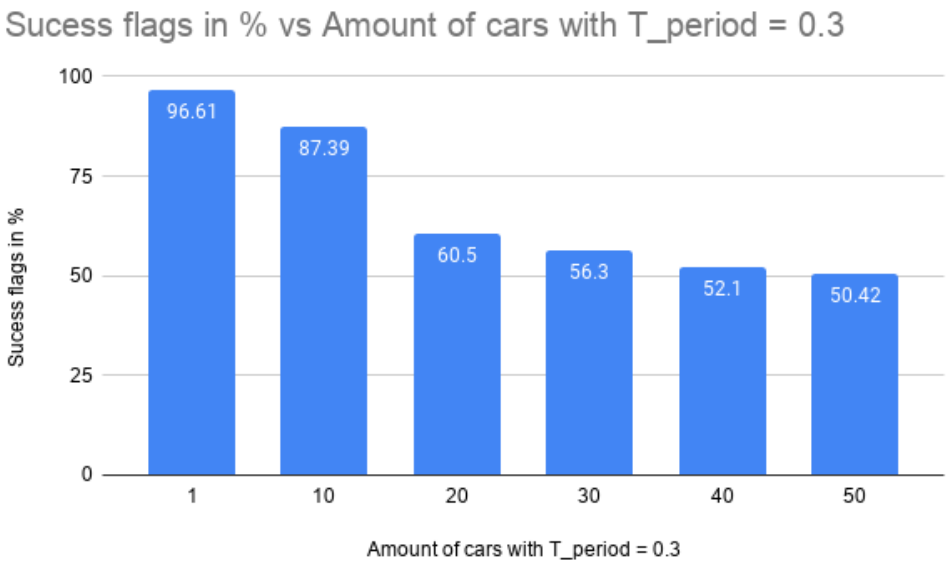
The used CCBS implementation is entirely single-threaded, so performance is mainly limited by the processing speed of the CPU core where the algorithm is executed. Therefore, the algorithm also could not benefit from using all eight available cores. The memory usage was mostly kept around 25 MB indicating that the lower level planner part of the CCBS was doing most of the work calculating new paths. This is because otherwise the memory usage would rise steadily when detecting new collisions creating more and more leaves in the constraint tree. Occasionally, the memory usage would spike upwards to 1 GB on the small map as a symptom of a large number of collisions between the agents, forcing the high-level planner to expand the constraint tree with equally many nodes, thereby demanding more use of the memory.

6.2 Collaborative Driving Management

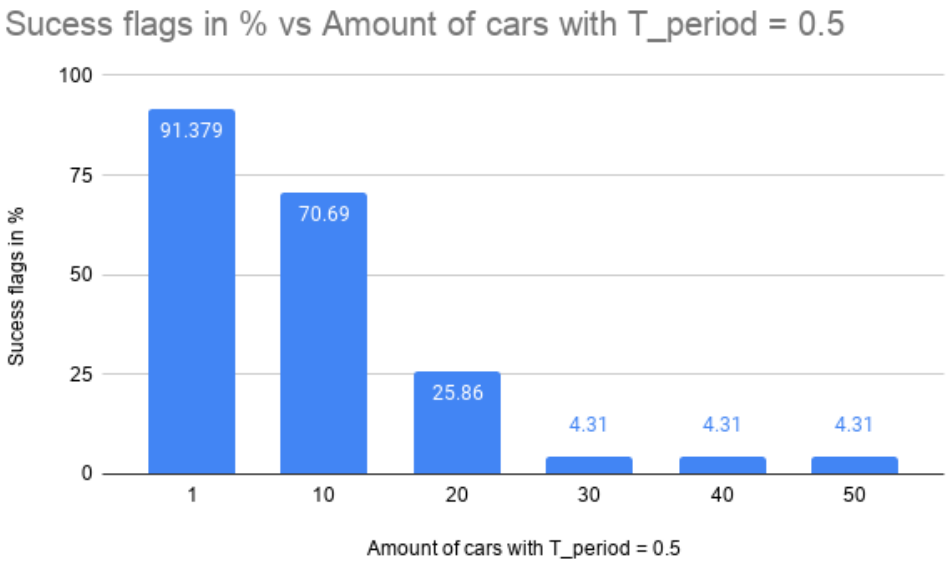
The tests on the frequency of the server and the number of vehicles were performed according to the methods described in Section 4.2.2. These tests show the relation between the number of cars involved in the intersection, the frequency of the server, and the intersection type. The results are demonstrated in the Fig. 6.6, 6.7 and 6.8.



(a) Frequency T_period = 0.1



(b) Frequency T_period = 0.3

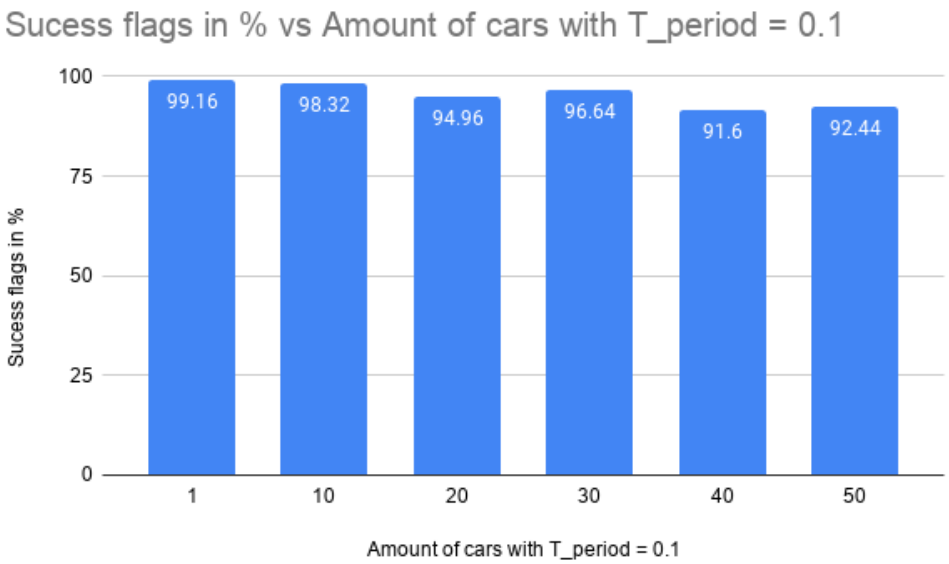


(c) FrequencyT_period = 0.5

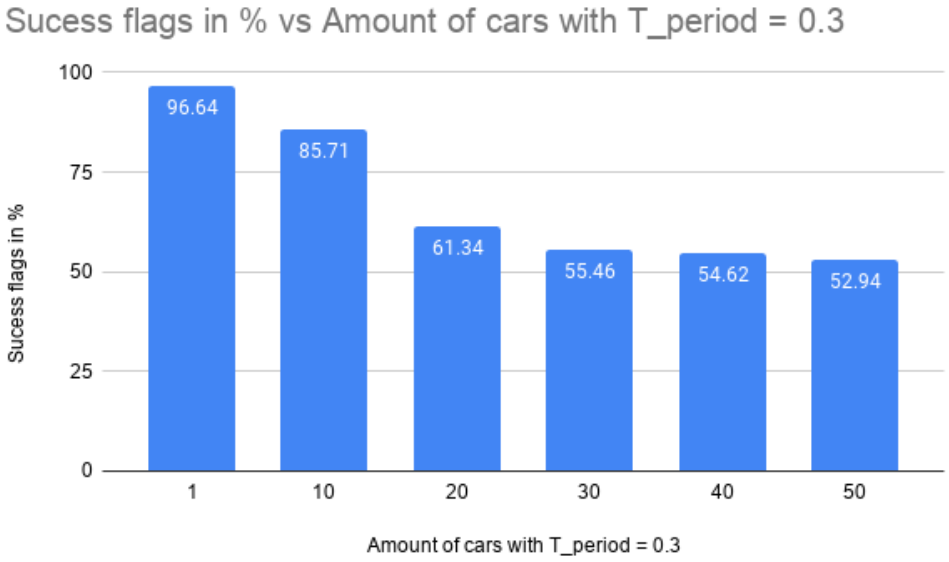
Fig. 6.6: 3-Way intersection: frequency of phase change; T_set = 0.5

In the 3-way roundabout the frequency level of 0.5 s the percentage decreases much

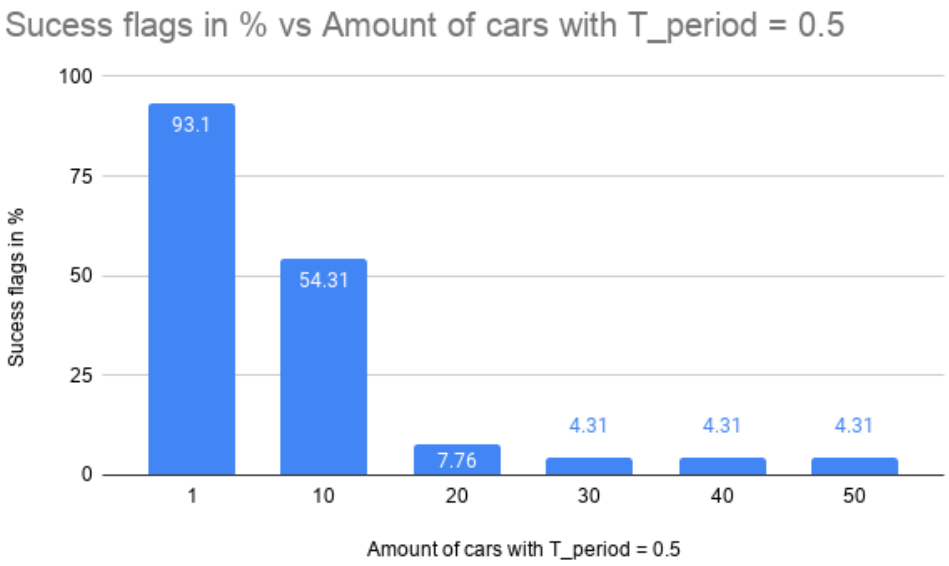
more in the compression to the 3-way intersection when the amount of car reaches the number 10 and 20:



(a) Frequency T_period = 0.1



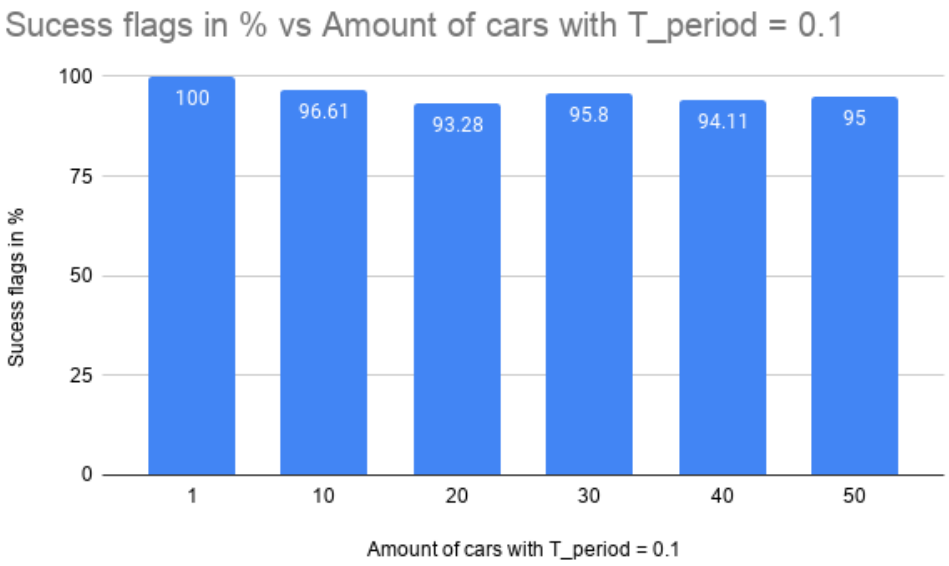
(b) Frequency T_period = 0.3



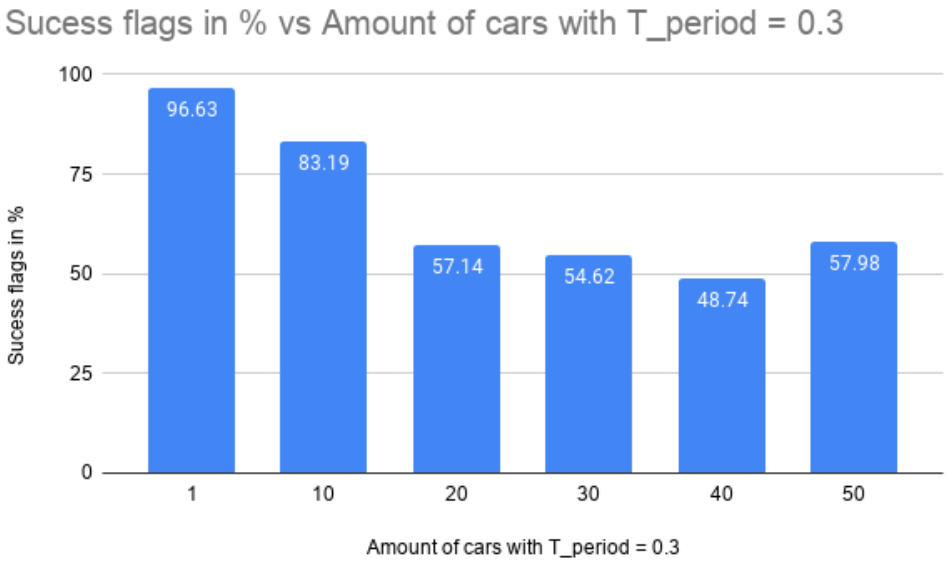
(c) Frequency T_period = 0.5

Fig. 6.7: 3-Way roundabout: frequency of phase change; T_set = 0.5

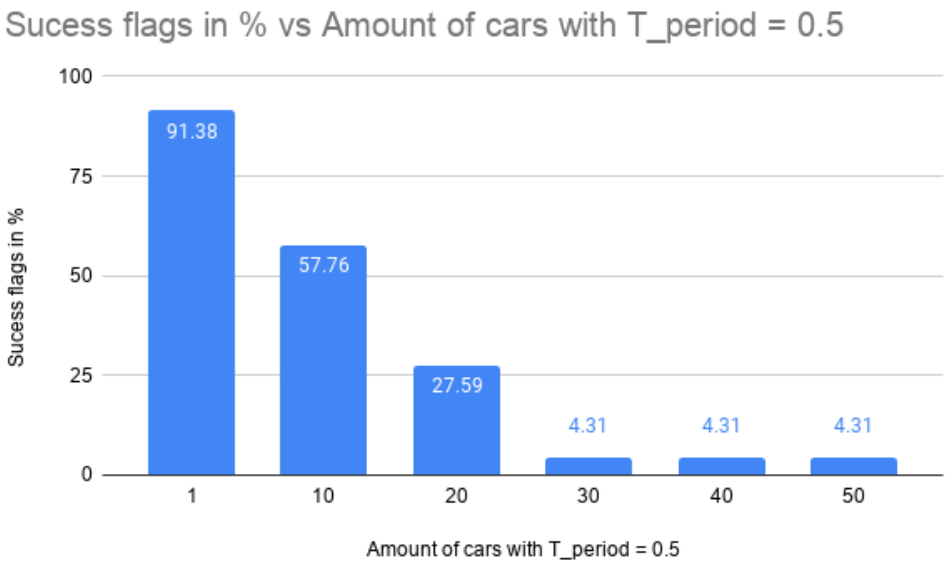
The 4-way test was the only test that ever reached a success percentage of 100 % in any amount of vehicle and T_period:



(a) Frequency T_period = 0.1



(b) Frequency T_period = 0.3



(c) Frequency T_period = 0.5

Fig. 6.8: 4-Way intersection: frequency of phase change; T_set = 0.5

It can be seen from the data presented in the graphs that the percentage of the agreement flag is very high for all types of critical sections and almost all the amount of cars when the period is 0.1 s. Even if more clients have been added, the decrease in agreements was not very significant.

When the period was 0.3 s the acceptance agreement was still very high for one vehicle, but after adding more clients this percentage decreased quicker than it had for a frequency value of 0.1. There is no big difference in the results when it comes to critical section type, but either way, the 3-way intersection has shown the highest results on percentage of the true agreement on the period 0.3 s, meanwhile 4-way intersection had the lowest percentage among all types. The 4-way intersection showed the highest rate of acceptance agreements on a period value of 0.1 s. Meanwhile the 3-way intersection and roundabout showed similar results, but with a marginally lower rate of agreements.

When the server's period is equivalent to the period of set activity (0.5 s), the percentage is still high for one vehicle, but the decrease in agreements is much more significant in comparison to the previous values of server's period. Especially, it can be seen when the number of clients reaches 30, where the acceptance is the same for every intersection at a low 4.31%. The roundabout has shown the lowest

result among all types of intersections with this period, at 7.76 % with 20 vehicles, meanwhile 3-way and 4-way intersections had 25 % and 27 % respectively. 3-way intersection has shown the highest average on the frequency 0.5, but when the number of clients reached 30 all critical sections showed the same result.

Chapter 7

Discussion

After implementing a prototype of the proposed system design and collecting quantitative data about its performance, this chapter discusses the results of the project, possible extendability, and improvements as well as social and ethical aspects related to this work.

7.1 Path-Planning for Many Agents

One of the main aims of this project was to implement a functioning multi-agent path planner. The CCBS algorithm had no issues finding collision-free routes for up to 100 agents on large scale maps, where the only limiting factor was the used model of the test map. Some irregularities in a few tests failing on the medium-sized map with 10 agents shown in Fig. 6.5 could be explained by the test map not being optimized to be used with the CCBS. For example, the euclidean distance between nodes is highly irregular in some sectors of the map, which causes unnecessary collisions when the node distance change from long to short distances. The map used in this project to simulate the entire system closely resembles the small map size used in the tests in terms of scale, where CCBS can plan routes for up to 20 agents reliably and swiftly. A limiting factor was the hardware on the test system. Using a faster processor would increase the performance dramatically allowing more agents to find routes.

Algorithms that can balance finding optimal routes with quick execution time and are reliable are hard to find. The results shown have proven that CCBS can handle a variety of road setups depending on the need of the system.

Python is an interpreted language, and unlike a compiled language like C++, Python is executed through an interpreter. While useful due to its simplicity and usage capability, this does come with a heavy penalty to runtime when compared with programs written in a compiled language. This section aims to show and compare execution times between the Python version of CCBS implemented in this project and a C++ version of CCBS [12].

C++/Python Runtime Comparison on Large Map			
Agents	**C++ (s)**	**Python (s)**	**Ratio**
10	0.0013	0.0594	45.6923
20	0.0027	0.1595	59.0741
30	0.0047	0.3143	66.8723
40	0.0071	0.5957	83.9014
50	0.0091	0.813	89.3407
60	0.0138	1.4058	101.8696
70	0.0197	2.2367	113.5381
80	0.0206	2.3684	114.9709
90	0.0356	4.7386	133.1067
100	0.0463	5.7622	124.4536
		Average ratio:	93.282

Table 7.1: Runtimes for Python and C++ on the large map as well as listing the ratio difference.

By referring to Table 7.1, the differences in runtimes between the C++ version and Python version on the large map (see Section 6.1) can be seen. C++ has an average runtime ratio of approximately 93, meaning that C++ is on average 93 times faster than the Python version.

The effects of this become more apparent during the reliability testing of the algorithm. As each test is given 30 seconds to succeed, tests exceeding this timeout are considered a failure and aborted. Given the runtime difference between the algorithm versions there are situations where the reliability of the C++ version becomes higher than the Python version.

Success Probability Comparison on Small Map		
Agents	**Probability C++**	**Probability Python**
10	1	1
15	1	1
20	1	1
25	0.96	0.96
30	0.92	0.76
35	0.6	0.4
40	0.56	0.2
45	0.2	0
50	0.04	0

Table 7.2: Reliability for Python and C++ on the small map.

Table 7.2 shows the average success rate for both versions of the algorithm on a small map. While the Python version quickly loses its reliability after 30 agents, C++ has a more stable decline until after 40 agents. The 30-second runtime mentioned before has a large part in the declining reliability of the algorithm. The inherent performance disadvantage of the Python version due to the nature of the language leads to lower reliability than the C++ version.

Giving a higher time limit would yield higher reliability values, but in the application in this project, 30 seconds was deemed to be a reasonable maximum wait time. In the case that the algorithm only calculates a new path for its agents once all agents are ready for one, all agents still have to wait until the slowest agent finishes its route until they can receive a new one and the time required to calculate a new route becomes travel time of slowest agent + calculation time. *Travel time of slowest agent* is provided by the algorithm already, although only approximately since there may be delays en route, for example introduced by the CDM system forcing a truck to make an unplanned stop.

As a final note, as seen in Table 7.1, the ratio is growing up until 100 agents. Due to the difficulties of adding support for more agents to the testing environment by adding more pockets to the map, it is difficult to know for certain the trend of the ratio after this point, but it seems reasonably safe to assume that the trend will continue.

7.2 Intersection Arbitration

One of the goals was to achieve safety in driving by implementing CDM inside the critical parts of the road. The architecture for the safety coordination system has been constructed in a modular and generalized way to create a potential for scalability.

7.2.1 Modularity and Generalization

During this project, a system for safe vehicle coordination in different critical sections has been developed. The design of this system is generalized and modularized, consist of a set of distributed servers, each of which coordinates movement in its seperate critical section. Such distributed structure is an optimal solution for a system where each critical section needs to be handled independently. This architecture is easy extendable and allows to add new types of critical sections without making big changes inside the system.

Another idea for implementing a modular system was considered initially. By implementing a centralized component to track all vehicle positions and perform presence checks for all connection zones in the system, instead of letting the vehicles do this, information management throughout the system would be simpler. Vehicles would no longer be required to store data about all critical sections and the addresses for all VTL servers, which would avoid transmitting and duplicating data in many places throughout the system. However, this design could prove fatally dangerous in case of communication issues, as vehicles would not be aware that they are approaching a critical section. Thus, it was decided to perform these checks on board every vehicle instead of using a cloud-based system.

7.2.2 Scalability of Critical Sections and Additional Lanes

The developed system considers four types of critical regions, out of which three sections are fully implemented, 3-way roundabout, 3- and 4-way intersections, and the final one, narrow road, whose implementation was not fully implemented in time. These four types represent a large portion of situations where traffic needs to be coordinated. However, the functionality here could be extended with more types of critical sections, primarily 4-way roundabouts, which is a common intersection type and bears a close resemblance to the already implemented 3-way roundabout and 4-way intersections.

The principle of using the non-conflicting sets in CDM allows the introduction of multiple driving lanes. The project uses one lane of travel for each direction in

all critical sections, but the system supports an extension to include two, three or even more This is because the system automatically calculates the non-conflicting sets based on the defined paths in the intersection file, and would function without any major adaptations.

7.2.3 Server Frequency

The experiments show that CDM is capable of handling up to 50 clients with a period of 0.1 s and set activity of 0.5 s without having a significant negative impact on the systems' ability to reach agreements. The results have also shown that the capacity of the server is not dependent on the type of critical section, except in cases where the frequency approached the set activity time. This means that the system with the relatively low frequency handles coordination properly in all implemented critical sections of the road.

The experiment has shown that the more difference between frequency time and set activity time, the more often the vehicles will accept the agreement. This behavior is due to how the server works. When T_period approaches T_set, the time to find a solution shrinks, therefore, reducing the percentage of agreements.

The most effective frequency was on T_period 0.1 and the 4-way intersection had the highest percentage of acceptance for this value. However, when the period of the server was increased up to 0.3 s, the 3-way intersection showed the highest results. As to the period level of 0.5 s, this number has shown to have the least number of true agreements. The high result achieved with one vehicle (around 91 to 93%) can be explained with that it initially fails several times but after the first successful agreement it will keep being successful until the test ends since no new vehicle is added.

7.2.4 Difficulties

One of the factors that slowed down the work progress in CDM development was poorly documented code that the group inherited to use for the VTL implementation and simulation. This created difficulties in understanding the concepts behind it and, therefore, caused more time to be spent on analyzing code. In this project, the CDM system has been commented and documented to make the system less complicated to use and develop further.

7.3 Social and Ethical Aspects

Though technical progress is something that is constantly chased, it is important to also consider what changes this would bring to society and the effects they would have.

7.3.1 Socioeconomic Aspects

Given that autonomous driving would become more incorporated in everyday life, it would affect many aspects of society, which is why there are currently huge investments in its development [1]. Professions such as truck, bus, or taxi drivers could become redundant, or at least the demand for labor within the field would decrease. It would probably lead to an increased demand within other fields of work such as the development of systems for autonomous vehicles. However, it could be considered problematic that a larger group of people with a lower education level would then become unemployed in exchange for a smaller group with a higher level of education getting work opportunities.

7.3.2 Ethical Aspects

Although the accidents caused by human errors would subside, autonomous systems come with other potential side effects [2]. Such a system could be attacked by malicious actors, which could lead to devastating consequences. Security would, therefore, be of utmost importance. There is also a matter of responsibility and liability should an accident occur. Even though accidents caused by factors like human fatigue, accidents would most likely happen, only in different situations. The autonomous driving systems will have to be programmed to make decisions, but this could be very difficult since there will not always be a right or wrong answer. An example of a situation where there is no clear answer is if a driver is going down a street and suddenly a child jumps out in the way chasing after a ball. The driver will not be able to stop in time and must steer right or left to not hit the child. But if he goes right he will hit a pedestrian, and if he goes left he will face a head-on collision with a car going too fast the opposite way. Though highly unlikely similar dilemmas would have to be accounted for when developing the systems. There would also be the question of who is liable, whether it be the production company, factories, software developers, or someone else. This liability could inhibit the incentive to work with the development of these products.

7.3.3 Environmental and Social Aspects

There is research that shows that autonomous vehicles have the potential to decrease greenhouse gas (GHG) emissions[5]. However, as the availability of these increases there is a possibility that the total travel will also increase, thus leveling out the reduction. It could even go as far as to increase the total emission of GHG. On the other hand, completely autonomous vehicles would allow groups that normally cannot travel alone or by public transport to do so. Such groups could include elderly people, people with disabilities, or people with no driver's license. It could also be helpful in global pandemics as there would be no need to put drivers in the public transport sector through the risk of getting infected, nor would it be a concern that there would be a lack of them.

7.4 Future Work and Extendability

The project resulted in an auto-driving system that has a possibility to be extended or to be a sub-component of a larger system. Therefore, there are a variety of possibilities for future work.

7.4.1 Simulation environment

The existing road map is suitable for testing a limited amount of vehicles since it consists of a confined road system with multiple intersections of various degrees and a roundabout. However to be able to test the system on a larger scale it would require a larger road system that has the capacity for more vehicles. To support an even more diverse road network, more types of intersections would need to be implemented as well as more road lanes versions. With the modularized design of CDM and versatility of non-conflicting sets, the task of adding new types of critical sections should not be that complex. Furthermore, the system uses four directions for road navigation to keep track of the vehicle entrances and exits; north, south, east, and west. To make the system more flexible the system could be redesigned to support directions in degrees instead. This would allow for a more diverse road network to be supported and make the system independent from the geographic location.

7.4.2 Expansion

The MAPF implemented in this project is limited by only utilizing one core a time. This could be expanded to support multi-core processors thus distributing the workload which would improve the performance dramatically. Especially in today's society when the number of cores per processor keeps expanding rapidly.

7.4.3 Trucks

For the simulation in the physical testbed there has been a maximum of two trucks running simultaneously. They had to be run from two different ROS instances. The new simulation system theoretically has support for an infinite number of cars. In practice however, when combined with the path-planner, CDM, and existing road map it is able to simulate up to a hundred vehicles. In the laboratory there are three truck models available. A possibility for future work is to integrate the new system with the physical trucks and get them to run in the testbed. For the sake of the optimality, dynamic speed could also be implemented.

Note that earlier efforts in the area of physical simulation included the Gulliver testbed, which is described in [29]–[31].

7.4.4 Safety

The designed CDM system currently has no fallback in case of a server failure. For the virtual traffic lights, safety would be maintained since cars would just stop at the intersection, but no cars would be able to cross since they will need a schedule. For the connection service, if there would have been any updates in an intersection, since last time the system updated its local database, such as changed connection address, the car could connect to the wrong server. This would be severe and could result in collisions.

There could be back up servers in a distributed system so if one goes down there is still at least one left running. With this you could also come around the danger of someone intentionally sabotage the server by not having them at the same location.

A different way to solve shut down of the virtual traffic light server would be that they resolve to a secondary vehicle to vehicle (V2V) protocol. By letting the vehicle, at a slow pace, safely coordinate themselves through the intersection.

An additional safety issue would be if a car breaks inside the intersection. As of now there is no control that all the cars that entered the intersection also exits. This could easily be implemented in the current system by counting the number of vehicles that enter and the number that exits, only if they are not the same within an interval would the intersections shut down and send an alert. This alert could be picked up by emergency services for quick response in case of an emergency and the multi-agent path finding service so it can redirect traffic.

Chapter 8

Conclusion

The project has resulted in a design for a planning and coordination system for autonomous vehicles, with of an optimal traffic planning algorithm and a coordination subsystem guaranteeing safety. The traffic planning algorithm is capable of planning routes for a fleet of vehicles in a closed road system. Traversal of critical sections is coordinated by CDM, which results in conflict-free driving which, in turn, achieves safety in the traffic. The combination of these two parts of the system makes it unique and applicable in different situations where autonomous driving is used and where both optimality and safety are important.

Bibliography

[1] LeasingOptions. (2018). The Race To Driverless Technology | Leasing Options, [Online]. Available: https://leasingoptions.co.uk/driverless-cars/index.html (visited on 04/28/2020).

[2] U. D. of Transportation, *Automated Driving Systems: A Vision for Safety*, Sep. 2017. [Online]. Available: https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/13069a-ads2.0_090617_v9a_tag.pdf (visited on 02/11/2020).

[3] J. Rhee and A. Valiente. (Sep. 18, 2014). The Danger of Forcing Truck Drivers to Drive Sleep-Deprived Exposed, [Online]. Available: https://abcnews.go.com/US/danger-forcing-truck-drivers-drive-sleep-deprived-exposed/story?id=25544862 (visited on 04/27/2020).

[4] European Commission. (2020). Driving time and rest periods, [Online]. Available: https://ec.europa.eu/transport/modes/road/social_provisions/driving_time_en (visited on 02/14/2020).

[5] Z. Wadud, D. MacKenzie, and P. Leiby, "Help or hindrance? The travel, energy and carbon impacts of highly automated vehicles", *Transportation Research Part A: Policy and Practice*, vol. 86, pp. 1–18, Apr. 2016, ISSN: 09658564. DOI: 10.1016/j.tra.2015.12.001. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0965856415002694 (visited on 04/28/2020).

[6] C. Linder. (Dec. 11, 2019). A Self-Driving Freight Truck Just Drove Across the Country to Deliver Butter, [Online]. Available: https://www.popularmechanics.com/technology/infrastructure/a30196644/self-driving-truck-cross-country/ (visited on 02/11/2020).

[7] E. Strid, I. Radjavi, J. Blom Rydell, O. Nilsson, V. Lyster, and W. Kruse, "Automation av ledade kommersiella tunga fordon", Chalmers Tekniska Högskola, Gothenburg, 2019, 36 pp.

[8] M. Andréasson and T. Hasselquist, "Arbitrating Intersection Crossing using V2I communications", Chalmers University of Technology, Gothenburg, Jun. 13, 2018, 54 pp.

[9] A. Casimiro, E. Ekenstedt, and E. M. Schiller, "Self-Stabilizing Manoeuvre Negotiation: The Case of Virtual Traffic Lights", in *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, Oct. 2019, pp. 354–3542. DOI: 10.1109/SRDS47363.2019.00048.

[10] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding", *Artificial Intelligence*, vol. 219, pp. 40–66, Feb. 1, 2015, ISSN: 0004-3702. DOI: 10.1016/j.artint.2014.11.006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370214001386 (visited on 04/21/2020).

[11] M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments", presented at the Proceedings - IEEE International Conference on Robotics and Automation, Jun. 13, 2011, pp. 5628–5635. DOI: 10.1109/ICRA.2011.5980306.

[12] A. Andreychuk, K. Yakovlev, D. Atzmon, and R. Stern, "Multi-Agent Pathfinding with Continuous Time", Jun. 14, 2019. arXiv: 1901.05506 [cs]. [Online]. Available: http://arxiv.org/abs/1901.05506 (visited on 04/21/2020).

[13] A. Casimiro, J. Kaiser, J. Karlsson, E. M. Schiller, P. Tsigas, P. Costa, J. Parizi, R. Johansson, and R. Librino, "Brief announcement: KARYON: Towards safety kernels for cooperative vehicular systems", in *Stabilization, Safety, and Security of Distributed Systems - 14th International Symposium, SSS 2012, Toronto, Canada, October 1-4, 2012. Proceedings*, A. W. Richa and C. Scheideler, Eds., ser. Lecture Notes in Computer Science, vol. 7596, Springer, 2012, pp. 232–235, ISBN: 978-3-642-33535-8. DOI: 10.1007/978-3-642-33536-5_22. [Online]. Available: https://doi.org/10.1007/978-3-642-33536-5_22.

[14] A. Casimiro, J. Kaiser, E. Schiller, P. Costa, J. Parizi, R. Johansson, and R. Librino, "The KARYON project: Predictable and safe coordination in cooperative vehicular systems", in *43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop, DSN Workshops 2013, Budapest, Hungary, June 24-27, 2013*, IEEE Computer Society, 2013, pp. 1–12, ISBN: 978-1-4799-0181-4. DOI: 10.1109/DSNW.2013.6615530. [Online]. Available: https://doi.org/10.1109/DSNW.2013.6615530.

[15] A. Casimiro, J. Rufino, R. C. Pinto, E. Vial, E. M. Schiller, O. M. Ponce, and T. Petig, "A kernel-based architecture for safe cooperative vehicular functions", in *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems, SIES 2014, Pisa, Italy, June 18-20, 2014*, IEEE,

2014, pp. 228–237. DOI: 10.1109/SIES.2014.6871208. [Online]. Available: https://doi.org/10.1109/SIES.2014.6871208.

[16] O. M. Ponce, E. M. Schiller, and P. Falcone, "Cooperation with disagreement correction in the presence of communication failures", in *17th International IEEE Conference on Intelligent Transportation Systems, ITSC 2014, Qingdao, China, October 8-11, 2014*, IEEE, 2014, pp. 1105–1110, ISBN: 978-1-4799-6078-1. DOI: 10.1109/ITSC.2014.6957835. [Online]. Available: https://doi.org/10.1109/ITSC.2014.6957835.

[17] A. Casimiro, O. M. Ponce, T. Petig, and E. M. Schiller, "Vehicular coordination via a safety kernel in the gulliver test-bed", in *34th International Conference on Distributed Computing Systems Workshops (ICDCS 2014 Workshops), Madrid, Spain, June 30 - July 3, 2014*, IEEE Computer Society, 2014, pp. 167–176, ISBN: 978-1-4799-4181-0. DOI: 10.1109/ICDCSW.2014.25. [Online]. Available: https://doi.org/10.1109/ICDCSW.2014.25.

[18] V. Savic, E. M. Schiller, and M. Papatriantafilou, "Distributed algorithm for collision avoidance at road intersections in the presence of communication failures", in *IEEE Intelligent Vehicles Symposium, IV 2017, Los Angeles, CA, USA, June 11-14, 2017*, IEEE, 2017, pp. 1005–1012. DOI: 10.1109/IVS.2017.7995846. [Online]. Available: https://doi.org/10.1109/IVS.2017.7995846.

[19] Thomas Petig, E. M. Schiller, and J. Suomela, "Changing lanes on a highway", in *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2018, August 23-24, 2018, Helsinki, Finland*, R. Borndörfer and S. Storandt, Eds., ser. OASICS, vol. 65, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 9:1–9:15. DOI: 10.4230/OASIcs.ATMOS.2018.9. [Online]. Available: https://doi.org/10.4230/OASIcs.ATMOS.2018.9.

[20] O. M. Ponce, E. M. Schiller, and P. Falcone, "How to stop disagreeing and start cooperatingin the presence of asymmetric packet loss", *Sensors*, vol. 18, no. 4, p. 1287, 2018. DOI: 10.3390/s18041287. [Online]. Available: https://doi.org/10.3390/s18041287.

[21] A. Casimiro, E. Ekenstedt, and E. M. Schiller, "Membership-based manoeuvre negotiation in autonomous and safety-critical vehicular systems", *CoRR*, vol. abs/1906.04703, 2019. arXiv: 1906.04703. [Online]. Available: http://arxiv.org/abs/1906.04703.

[22] J. Yu and S. M. LaValle, "Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs", in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, Jun. 29, 2013. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6111 (visited on 04/20/2020).

[23] T. S. Standley, "Finding Optimal Solutions to Cooperative Pathfinding Problems", in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, Jul. 3, 2010. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1926 (visited on 04/22/2020).

[24] C. Wuthishuwong, A. Traechtler, and T. Bruns, "Safe trajectory planning for autonomous intersection management by using vehicle to infrastructure communication", *EURASIP Journal on Wireless Communications and Networking*, vol. 2015, no. 1, p. 33, Dec. 2015, ISSN: 1687-1499. DOI: 10.1186/s13638-015-0243-3. [Online]. Available: https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-015-0243-3 (visited on 06/03/2020).

[25] K. Dresner and P. Stone, "Multiagent traffic management: A reservation-based intersection control mechanism.", vol. 2, Jan. 2004, pp. 530–537. DOI: 10.1109/AAMAS.2004.190.

[26] ROS Wiki Contributors. (Aug. 8, 2018). ROS/Introduction - ROS Wiki, [Online]. Available: http://wiki.ros.org/ROS/Introduction (visited on 04/29/2020).

[27] ——, (Aug. 14, 2015). Rviz UserGuide - ROS Wiki, [Online]. Available: http://wiki.ros.org/rviz/UserGuide (visited on 04/29/2020).

[28] ——, (Mar. 5, 2019). Tf2 - ROS Wiki, [Online]. Available: http://wiki.ros.org/tf2 (visited on 05/12/2020).

[29] M. Pahlavan, M. Papatriantafilou, and E. M. Schiller, "Gulliver: A test-bed for developing, demonstrating and prototyping vehicular systems", in *Proceedings of the 75th IEEE Vehicular Technology Conference, VTC Spring 2012, Yokohama, Japan, May 6-9, 2012*, IEEE, 2012, pp. 1–2, ISBN: 978-1-4673-0989-9. DOI: 10.1109/VETECS.2012.6239951. [Online]. Available: https://doi.org/10.1109/VETECS.2012.6239951.

[30] C. Berger, E. Dahlgren, J. Grunden, D. Gunnarsson, N. Holtryd, A. Khazal, M. Mustafa, M. Papatriantafilou, E. M. Schiller, C. Steup, V. Swantesson, and P. Tsigas, "Bridging physical and digital traffic system simulations with the gulliver test-bed", in *Communication Technologies for Vehicles, 5th International Workshop, Nets4Cars/Nets4Trains 2013, Villeneuve d'Ascq, France, May 14-15, 2013. Proceedings*, M. Berbineau, M. Jonsson, J.-M. Bonnin, S. Cherkaoui, M. Aguado, C. R. Garcia, H. Ghannoum, R. Mehmood, and A. V. Vinel, Eds., ser. Lecture Notes in Computer Science, vol. 7865, Springer, 2013, pp. 169–184, ISBN: 978-3-642-37973-4. DOI: 10.1007/978-3-642-37974-1_14. [Online]. Available: https://doi.org/10.1007/978-3-642-37974-1_14.

[31] C. Berger, O. M. Ponce, T. Petig, and E. M. Schiller, "Driving with confidence: Local dynamic maps that provide LoS for the gulliver test-bed",

in *Computer Safety, Reliability, and Security - SAFECOMP 2014 Workshops: ASCoMS, DECSoS, DEVVARTS, ISSE, ReSA4CI, SASSUR. Florence, Italy, September 8-9, 2014. Proceedings*, A. Bondavalli, A. Ceccarelli, and F. Ortmeier, Eds., ser. Lecture Notes in Computer Science, vol. 8696, Springer, 2014, pp. 36–45. DOI: 10.1007/978-3-319-10557-4_6. [Online]. Available: https://doi.org/10.1007/978-3-319-10557-4_6.