HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Matthias Artmann

Simulative Performance Evaluation of a Provably Collision Free Approach for Managing Autonomous Cars on Urban Intersections

Bachelor Thesis in Computer Science

14 May 2020

C M S

Cooperative Mobile Systems
Heinz Nixdorf Institute, Paderborn University, Germany

Fürstenallee 11 · 33102 Paderborn · Germany

http://www.cms-labs.org/

Simulative Performance Evaluation of a Provably Collision Free Approach for Managing Autonomous Cars on Urban Intersections

Bachelor Thesis in Computer Science

vorgelegt von

Matthias Artmann

geb. am 09. November 1997
in Paderborn

angefertigt in der Fachgruppe

Cooperative Mobile Systems

Heinz Nixdorf Institut
Universität Paderborn

Betreuer:	**Christoph Sommer**
Gutachter:	**Christoph Sommer**
	Friedhelm Meyer auf der Heide
Abgabe der Arbeit:	**14. Mai 2020**

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.
Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Matthias Artmann)
Paderborn, 14 May 2020

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Eidesstattliche Versicherung

Nachname	Artmann	Vorname	Matthias
Matrikelnr.	7087806	Studiengang	Informatik

[✓ Bachelorarbeit] Masterarbeit

Titel der Arbeit: Simulative Performance Evaluation of a Provably Collision Free Approach for Managing Autonomous Cars on Urban Intersections

☐ Die elektronische Fassung ist der Abschlussarbeit beigefügt.

☒ Die elektronische Fassung sende ich an die/den erste/n Prüfenden bzw. habe ich an die/den erste/n Prüfenden gesendet.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit (Ausarbeitung inkl. Tabellen, Zeichnungen, etc.) selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Abschlussarbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Die elektronische Fassung entspricht der gedruckten und gebundenen Fassung.

Belehrung

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist die Vizepräsidentin / der Vizepräsident für Wirtschafts- und Personalverwaltung der Universität Paderborn. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz NRW in der aktuellen Fassung).

Die Universität Paderborn wird ggf. eine elektronische Überprüfung der Abschlussarbeit durchführen, um eine Täuschung festzustellen.

Ich habe die oben genannten Belehrungen gelesen und verstanden und bestätige dieses mit meiner Unterschrift.

Ort	Paderborn	Datum	14.05.2020

Unterschrift [illegible]

Abstract

In the development of intelligent, autonomously operating vehicles, safety is a vital property every system design must ensure in order to be considered for real applications. By enabling autonomous vehicles to communicate with each other, they gain the opportunity to coordinate their maneuvers and thereby improve their safety cooperatively. A recent approach to specify vehicle controllers for coordinated crossing maneuvers at urban intersections, presented by Schwammberger [1], uses a mathematical formalization of the problem to prove the collision freedom of an exemplary controller. Because the formalism employs a highly abstract traffic model, it is unclear how such a controller should be implemented and to what extent its safety properties translate into a more realistic scenario.

In this thesis, I investigate these questions by implementing the controller in the vehicular network simulation framework Veins and conducting extensive simulation studies. I define simulation parameters that introduce realistic communication and coordination restrictions and I evaluate the controller's performance in terms of vehicle safety. The simulation results show that the controller preserves its collision freedom in the least restricted configurations, confirming that in principal, the safety property translates to more realistic scenarios. For configurations with more severe restrictions, however, the coordination fails and vehicles can collide.

By analyzing the reasons for the coordination failures, I conclude that the example controller alone is not suitable for more realistic environments in which the simplifying assumptions of the abstract traffic model do not hold. However, the controller can most likely be adjusted to handle more realistic environments by using a more sophisticated communication protocol for the implementation or by extending its abstract definition directly, which is explicitly supported by the abstract model because it can be used to specify arbitrary controllers.

Kurzfassung

Bei der Entwicklung intelligenter autonom operierender Fahrzeuge ist die Sicherheit eine unerlässliche Eigenschaft, die jedes System erfüllen muss, um für reale Anwendungen in Betracht gezogen zu werden. Indem man es den Fahrzeugen ermöglicht, miteinander zu kommunizieren, gibt man ihnen die Möglichkeit, ihre Manöver zu koordinieren und somit ihre Sicherheit kooperativ zu verbessern. Ein kürzlich von Schwammberger [1] vorgestellter Ansatz, Fahrzeugsteuerungen für koordinierte Kreuzungsmanöver auf städtischen Kreuzungen zu definieren, verwendet eine mathematische Formalisierung des Problems, um die Kollisionsfreiheit einer beispielhaften Steuerung zu beweisen. Da der Formalismus ein hochgradig abstraktes Verkehrsmodell verwendet, ist es unklar, wie eine solche Fahrzeugsteuerung implementiert werden sollte und inwiefern sich ihre Sicherheitseigenschaften auf ein realistischeres Szenario übertragen.

In dieser Arbeit gehe ich diesen Fragen nach, indem ich die Steuerung im Simulationsframework Veins für vernetzte Fahrzeuge implementiere und umfangreiche Simulationsstudien durchführe. Ich definiere Simulationsparameter, die die Kommunikation und Koordination auf realistische Weise einschränken, und ich bewerte die Leistung der Fahrzeugsteuerung anhand der Sicherheit der Fahrzeuge. Die Simulationsergebnisse zeigen, dass die Steuerung ihre Kollisionsfreiheit in den am wenigsten einschränkenden Konfigurationen beibehält, was bestätigt, dass sich die Sicherheitseigenschaften prinzipiell auf realistischere Szenarien übertragen lassen. Für Konfigurationen mit stärkeren Einschränkungen schlägt die Koordination allerdings fehl und es kann zu Kollisionen zwischen den Fahrzeugen kommen.

Indem ich die Gründe für die fehlschlagende Koordination analysiere, komme ich zu dem Schluss, dass die beispielhafte Steuerung allein nicht für realistischere Szenarien, in denen die vereinfachenden Annahmen des abstrakten Modells nicht gelten, geeignet ist. Allerdings kann die Steuerung höchstwahrscheinlich angepasst werden, um auch in realistischeren Umgebungen zu funktionieren, indem man für die Implementierung ein weiter entwickeltes Kommunikationsprotokoll verwendet, oder indem man ihre abstrakte Definition direkt erweitert. Dies wird vom abstrakten Mo-

dell explizit unterstützt, da man es für die Definition beliebiger Fahrzeugsteuerungen verwenden kann.

Contents

Abstract **iv**

Kurzfassung **v**

1 Introduction **1**

1.1 Related Work 3

2 Fundamentals **6**

2.1 Abstract Urban Traffic Model 6

2.2 Crossing Controller 13

2.3 Simulation Framework 16

3 Controller Implementation **20**

3.1 Controller Design Considerations 20

3.2 Traffic Model Interface 23

3.3 Vehicle Controller Application 32

3.4 Practical Implementation Problems 38

3.5 Verification and Validation 42

4 Simulation Setup **46**

4.1 Intersection Scenarios 46

4.2 Traffic Generation 48

4.3 Simulation Parameters 51

4.4 Performance Metrics 56

5 Evaluation **61**

5.1 Implementation and Abstraction Level 61

5.2 Simulation Results 63

6 Conclusion **96**

Bibliography **102**

Chapter 1

Introduction

Autonomously operating vehicles offer great potential for improving numerous aspects of future transportation, such as safety, traffic flow efficiency, and ecological sustainability. By constructing Vehicular Ad Hoc Networks (VANETs) via means of short-range wireless communication, traffic agents are able to cooperatively solve complex coordination and optimization problems. For instance, a very active area of research investigates the use of Cooperative Adaptive Cruise Control (CACC) for *vehicular platooning*, where multiple vehicles cooperate to drive at a minimum safety distance to the preceding and following vehicle and thereby optimize their total air resistance and road usage [2]. The major benefit of this approach is improved traffic efficiency in terms of reduced fuel consumption and increased road capacity.

One task essential to the success of autonomous mobility concepts is to guarantee the safety of all participants in any situation. The scenario of urban intersections is especially interesting in this regard due to the multitude of interfering driving directions and the highly heterogeneous street layout, in comparison to motorways or country roads.

In recent approaches to solving this problem using methods of theoretical computer science, mathematical formalisms for describing and reasoning about traffic situations from a high-level abstract perspective were introduced and used to implement provably collision-free vehicle controllers for specific traffic scenarios. Building on *Multi-lane Spatial Logic (MLSL)*, introduced by Hilscher et al. [3] for a multi-lane motorway scenario, Hilscher and Schwammberger [4] proposed an extension of MLSL to *Urban Multi-lane Spatial Logic (UMLSL)* to cover urban traffic scenarios with intersections. A more refined version of UMLSL was later presented by Schwammberger [1], extending the formal model and vehicle controller to support intersections of arbitrary size. Another extension of the controller, also based on the original version [4], introduced a communication protocol to deal with imperfect knowledge scenarios [5].

Approaching the highly complex problem of coordinating maneuvers of cooperative autonomous vehicles from a high level of abstraction allows scientists and engineers to develop and evaluate new control strategies quickly and independently of specific vehicle or road network characteristics. The additional possibility to formally prove that a controller design satisfies the imposed safety and efficiency requirements further motivates the development of suitable mathematical formalisms. However, it is unclear how well such an abstract control mechanism and its formal properties translate into a realistic environment and perform in the presence of influences that are not captured by the formalism.

Although both mentioned extensions of the UMLSL-based approach [4] modify the vehicle controller to deal with somewhat more realistic scenarios [1], [5], both versions still rely on strong idealizing assumptions to make their safety proofs possible. Therefore, the controller cannot be expected to provide the same level of guaranteed safety under more realistic conditions. This raises the question to what extent the safety properties of the crossing controller are affected by (partially) lifting the idealizing assumptions. It is also unclear how this controller that is based on an abstract, purely spatial formalism can be implemented in software in a sensible and practicable manner.

In this thesis, I take a simulative approach to investigate both of these questions. To this end, I develop a suitable implementation of the crossing controller for arbitrary intersections [1] as an on-board application in the vehicular simulation framework *Veins* [6]. The framework provides realistic communication models that accurately simulate wireless channel characteristics, and a microscopic traffic simulator that can be used to investigate a wide range of traffic scenarios. I document the implementation process in this environment and report on the major design challenges. Thereafter, I use the implemented application for extensive simulation studies to examine the controller's performance in various intersection scenarios. Here, I investigate the influence that several parameters introducing an increasing degree of realism have on the safety of crossing maneuvers. Because the vehicle controller implements a cooperative coordination policy that requires the vehicles to exchange information with each other, and since inter-vehicular communication (IVC) is one of the most challenging research topics in this domain [7], I focus on realistic communication models and artificial communication impairments as the main parameters. Especially the impact of limited transmission range and reliability is of interest in the urban intersection scenario due to buildings and other obstacles blocking the line of sight between vehicles approaching the intersection from different directions. Additionally, I introduce a message processing delay and random errors in the perception of other vehicles' positions to model limitations and imperfections of the on-board hardware and examine their effects on the crossing maneuver safety.

1.1 Related Work

In the context of the growing relevance of intelligent, autonomous vehicles in recent years, there exists a substantial amount of research dealing with numerous challenges in all facets of this domain. The urban scenario and intelligent intersection management in particular have proven to be demanding areas and various approaches to address their issues using means of IVC and formal modeling have been developed [8]–[10].

Dresner and Stone [11] proposed a reservation-based intersection management system for autonomous vehicles with the goal of optimizing intersection throughput by modeling the coordination problem as a system of cooperating agents. Their approach is based on splitting the intersection's area into a grid of $n \times n$ *reservation tiles*. Every vehicle that plans to cross the intersection must request a slot in time and space, expressed as a time interval for each tile it requires. A central intersection controller responds to all requests and determines whether to accept or reject them by ensuring that no tile is reserved by more than one vehicle at the same time. A First Come First Serve policy is employed to determine which vehicles are accepted first. The reservation system was implemented in a custom simulation environment and compared to a traffic light and an overpass, the optimal solution in this scenario, for several intersection sizes and granularity values n. The results show a significant improvement in average delay over the traffic light policy, approaching the performance of the overpass.

In later elaborations of this system, more detailed vehicle characteristics, a sophisticated communication protocol, and support for mixed traffic consisting of fully autonomous and human-controlled vehicles were implemented [12]. The advantages of this reservation-based approach over traditional intersection management systems in terms of efficiency were evaluated thoroughly by simulation studies in several four-way intersection scenarios. Additionally, the communication protocol was designed to be robust against message loss to provide a high level of safety, even under extremely unreliable communication conditions. The intersection manager was supplemented by means to counter effects of imperfect vehicle sensors and actuators as well as detect crashed vehicles on the intersection and prevent approaching vehicles from entering.

Another approach was taken by Kowshik, Caveney, and Kumar [13], who designed a hybrid system of a centralized intersection management unit and a distributed safety control mechanism. In their system, a central intersection manager is responsible for assigning a time slot to each approaching vehicle during which it may perform its crossing maneuver. The time slots are selected in such a way that slots for conflicting maneuvers are always disjoint. However, the vehicles must ensure on their own that they can adhere to their time slots while keeping a safe distance to each other, relying on periodically updated information on the leading car's speed and acceleration. By assuming worst case behavior and acting cautiously, the control policy can deal with noisy sensor information and delayed or lost coordination messages, given that maximum error bounds are known.

The authors designed formalisms for all components of their model, which allowed them to prove the safety and liveness of their approach. Using a degree of freedom in the intersection manager's time slot assignment policy, they also improved the efficiency of their system, which they evaluated for a simulated four-way junction with a single lane per driving direction. The results showed a significant reduction of the average travel time compared to a traffic light and a stop sign policy for traffic loads of up to two vehicles per second.

A more practically oriented application was presented by Hafner et al. [14]. Instead of relying on fully autonomous driving for cooperative maneuver coordination, they focused on collision avoidance by intervention, i.e., their control system only becomes active in case it detects a potentially dangerous situation. The detection mechanism is based on a simplified model of the vehicle dynamics and uses a *capture set* to describe configurations of vehicle positions and speeds for which no acceleration or brake input can prevent a collision if the vehicles continue following their current trajectories. It is defined to only consider two vehicles at a time. The vehicles constantly exchange state information to allow the collision avoidance system to detect when a configuration in the capture set is approached. As soon as this happens, the system calculates control inputs for both vehicles in order to prevent a collision.

The authors implemented this application in two test vehicles equipped with the necessary communication and computing hardware, and evaluated its functionality experimentally, finding that it can successfully avert collisions in a real-world scenario. They repeated the experiment later with an improved formal model to deal with inaccurate sensor information and communication delay, showing similar results [15].

In an attempt to leverage vehicle platooning to improve the efficiency of coordination-based intersection management, Bashiri and Fleming [16] designed a novel coordination policy and compared it to a common stop sign method. Their policy uses a central intersection management unit that receives crossing requests from the leading vehicles of platoons approaching the intersection. The request messages include information on the platoon's size, speed, and position that allows the management unit to apply a scheduling algorithm to minimize the average waiting time or the waiting time's variance. Using a simulation environment implemented in Matlab, the authors were able to show that their approach significantly outperforms the regular stop sign policy in terms of waiting time and reduces the communication overhead in comparison to a solution without platooning.

Later, Bashiri, Jafarzadeh, and Fleming [17] developed this system further by adding more realistic vehicle dynamics and optimizing their scheduling techniques. Also by running simulations, they compared the improved system to a traffic light policy with fixed phases and found that it can increase the intersection's throughput while reducing average waiting times and fuel consumption.

It is evident that cooperative intersection management and especially its safety requirements are a highly active area of research. The majority of coordination approaches are based on simplifying formal models that are used to reduce the management problem to scheduling and optimization tasks. These models are frequently used to prove safety and liveness properties, while the efficiency of the control strategies is often evaluated using simulations or real-world experiments. Most of the presented approaches consider the coordination problem at multiple abstraction levels, e.g. by introducing high-level formalisms to describe their reasoning and control policies, and implementing their systems in microscopic traffic simulations with detailed vehicle dynamics and communication protocols.

The performance evaluation on a low abstraction level considering a concrete implementation of a control system in the presence of imperfect information and communication is essential for the validation of a system's safety properties. However, the abstract intersection coordination protocol presented by Schwammberger [1] is only defined at a very high level of abstraction and has not been evaluated in this way yet.

Chapter 2

Fundamentals

The abstract traffic model and vehicle controller presented by Schwammberger [1] are the main subject of this thesis. Therefore, it is essential to provide sufficient information on how they work and what concepts and techniques are used to define them. Their evolutionary development process is spread over several years and publications [1], [3]–[5], [18], addressing various application scenarios and iteratively extending the model to cope with ever more complex traffic environments. This chapter discusses only a selection of topics in depth and provides superficial explanations of the remaining aspects, ensuring an intuitive understanding.

To this end, the basic concepts of the abstract urban traffic model and UMLSL are addressed in Section 2.1, followed by a simplified definition and explanation of the vehicle controller in Section 2.2.

Section 2.3 provides an overview of the simulation framework Veins. It pays special attention to the traffic simulator SUMO and the differences between its traffic representation and the abstract model, which are essential for the implementation of the vehicle controller.

2.1 Abstract Urban Traffic Model

The abstract traffic model was first introduced by Hilscher et al. [3], only describing multi-lane motorways, and incrementally extended to country roads with two-way traffic [18], intersections of two roads with two lanes each [4], and, finally, intersections of roads with arbitrary lanes [1]. This last stage of development is the basis of the thesis and subject of this section.

The model comprises two main components: A generic, mathematical formalism to abstractly describe the topology of urban road networks as well as the vehicles inside, and a formal logic that can be used to reason about traffic situations, called Urban Multi-lane Spatial Logic (UMLSL) [1]. On top of this model, a vehicle controller for safe intersection crossing maneuvers is defined. Using UMLSL to reason about its behavior, it was shown that if all vehicles in a traffic network are equipped with this controller, no collisions can occur.

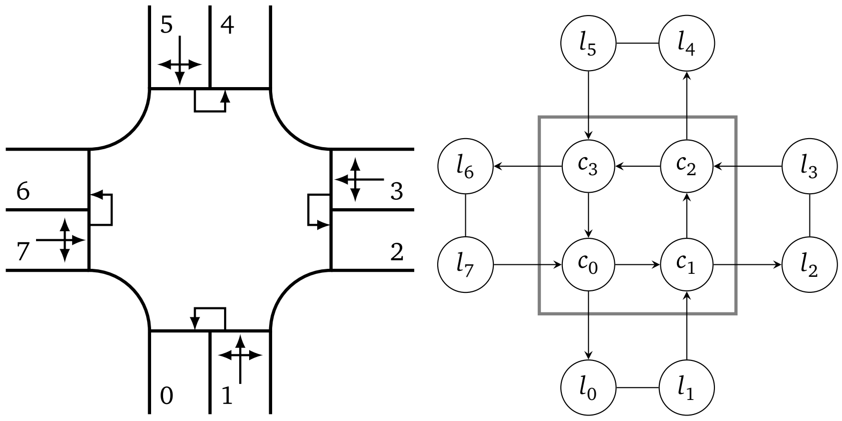
2.1.1 Road Network and Traffic Representation

The abstract traffic model describes urban road networks as graphs with directed and undirected edges, where the nodes are either *lane segments* or *crossing segments*. Each lane segment represents one lane that connects two intersections and each crossing segment represents a certain part of an intersection. Lane segments that belong to the same road, i.e., connect the same intersections, are connected by undirected edges that define a neighborhood relation; lane changes are only possible between lanes connected by these edges. Neighboring lanes may have opposing driving directions, allowing vehicles to perform overtaking maneuvers by temporarily using a lane of the oncoming traffic.

Crossing segments are only connected to other nodes by directed edges. An edge between a crossing segment c and a lane segment l means that the intersection to which c belongs can be entered from, or left via, the lane l, depending on the direction of the edge. Edges between crossing segments are mainly used to describe possible paths for crossing maneuvers. Apart from that, each maximal subgraph of crossing segments, ignoring the direction of the edges, identifies and describes exactly one intersection in the road network. Figure 2.1 illustrates this structure for a simple intersection connected to four roads, each having one lane leading towards the intersection and one leading away from it.

Intersections like this with n lanes on each of the four roads are called $n \times n$ intersections from here on; using this notation, the example shows a 2×2 intersection. This can be generalized to $n \times m$ intersections for the case where only roads on opposing sides of the intersection have equal numbers of lanes: By convention, n is the number of lanes on the two horizontal roads and m is the number of lanes on the vertical roads; the numbers of incoming lanes per road are $\frac{n}{2}$ and $\frac{m}{2}$ respectively.

Every segment s in the network has a size $\omega(s) \in \mathbb{R}^+$. For lane segments, the size directly corresponds to the length of the represented lane. A position on lane l is a real number $p \in [0, \omega(l)]$. The driving direction of l is then defined as the direction of increasing position value. The size of a crossing segment does not explicitly describe the shape of the segment; instead, it is used to calculate the length of crossing maneuver trajectories that incorporate the segment. Accordingly, a position $p \in [0, \omega(c)]$ on a crossing segment c can only be interpreted in relation to a maneuver. The traffic model defines a number of sanity conditions for the network structure to ensure that there are no dead ends, self loops of edges, or other



(a) Detailed intersection layout **(b)** Abstract graph representation

Figure 2.1 – Intersection representation in the abstract traffic model. Figure 2.1a shows the layout and shape of a 2×2 intersection in detail. A possible graph representation of this intersection is shown in Figure 2.1b. The nodes labeled with c_i are crossing segments and the l_i are lane segments. The maximal subgraph of crossing segments marked by the gray box is the representative of the whole intersection. Its depiction is based on [1, Figure 2].

inconsistencies, and that driving directions of lanes are well defined. However, their exact formalization is of no interest here.

Vehicles driving in the network are uniquely identified by capital letters $A, B, C, \ldots$. Typically, variables $a, b, c, \ldots$ are used for describing arbitrary vehicles. The *vehicle under consideration* is conventionally referred to as *ego*, which can be seen as a special vehicle variable. The position description of a vehicle c comprises a lane or crossing segment s and a real-valued position p on that segment, where p is interpreted as the position of the vehicle's rear bumper. Every vehicle follows an individual route that is represented by an infinite sequence of segments called *infinite path* $pth(c)$, where each successive pair of segments (s_i, s_{i+1}) is connected by a directed or undirected edge. A crossing maneuver in this path is simply defined as a partial sequence of contiguous crossing segments delimited by the two lane segments through which the vehicle enters and leaves the intersection.

The area occupied by a vehicle is called its *safety envelope* and contains the vehicle's physical size as well as its current braking distance added in front of the front bumper. Since lane segments have no notion of width, the safety envelope is defined to span the whole width of the lane represented by the current lane segment. Crossing segments, on the other hand, are handled differently: If a vehicle's safety envelope extends to a crossing segment, the whole segment is assumed to be occupied by the corresponding vehicle. This is important for the coordination mechanics used by the vehicle controller.

The abstract traffic model defines a data structure called *Traffic Snapshot* that stores information on all vehicles at a given point in time. For example, given a vehicle C, the Traffic Snapshot of the current time provides the infinite path $pth(C)$, the index of the current segment $curr(C)$ such that $s = pth(C)(curr(C))$ is the current segment, and the position $pos(C)$ on the segment s. Note that this only describes the position of the vehicle's rear bumper. The safety envelope typically extends over several other segments that follow s in $pth(C)$.

The passing of time is modeled by so-called *time transitions*. A time transition for a given duration $t \in \mathbb{R}^+$ can be applied to a Traffic Snapshot to obtain another Traffic Snapshot where the information on all vehicles is updated to reflect the situation t time units in the future. Other transition types model *actions* performed by the vehicles and do not progress time; vehicles can, for instance, change their acceleration instantaneously. Apart from the vehicle controller and a number of sanity conditions, the abstract model does not define rules or even an execution model to describe when transitions should be applied.

For the purpose of coordination, the abstract model defines the concepts of *claims* and *reservations*. Formally, both claims and reservations are sets of lane or crossing segments. The abstract model draws a distinction between lane and crossing segments but for understanding the coordination technique, it suffices to only address claims and reservations of crossing segments. In a Traffic Snapshot, one set of claimed segments $cclm(c)$ and one set of reserved segments $cres(c)$ is stored for each vehicle c. Intuitively, the reserved segments of a vehicle are those segments that it currently occupies and on which no other vehicle should be driving. The claimed segments are those segments that the vehicle plans to reserve in the near future.

Vehicles can modify their claims and reservations by performing the appropriate *actions*. The relevant actions are

- `cc`: Claim all crossing segments that are part of the next crossing maneuver
- `wd_cc`: Withdraw all claims on crossing segments, i.e., empty the set $cclm(\text{ego})$
- `rc`: Reserve all crossing segments that are currently claimed and withdraw those claims; effectively moves segments from $cclm(\text{ego})$ to $cres(\text{ego})$
- `wd_rc`: Withdraw all reservations on crossing segments, i.e., empty the set $cres(\text{ego})$

The abstract model differentiates between the actions performed by the vehicle controller and the transitions that are subsequently applied to Traffic Snapshots. Again, it suffices here to regard them as one and the same.

Finally, the abstract traffic model defines the concept of *views* to describe how vehicles perceive their surroundings. A view is constructed for a certain vehicle at a given moment in time, consists of several lane and crossing segments, and is bounded by a *horizon* $h \in \mathbb{R}^+$ that can be interpreted as the vehicle's view distance. In simple terms, a view follows the route of the vehicle, with and against its driving direction, as far as the horizon h allows it. Its structure can be compared to that of a safety envelope that stretches over all lanes belonging to the same road and that follows multiple intersection crossing maneuvers simultaneously. The formal definition of views and their construction is highly complex and involves several intermediate constructs that are used to assemble a sufficiently detailed final view structure. For simplicity, the notation $V(c)$ is used here to denote the entire area a vehicle c can perceive, without revealing details of the actual view data structure. In addition to their view, vehicles also have limited access to the current Traffic Snapshot. What this means becomes clear when looking at how views are utilized.

One of the main purposes of views is the perception of nearby vehicles and, more importantly, their safety envelopes, claims, and reservations. To this end, every vehicle is assumed to be equipped with sensors that are capable of perceiving the full length of other vehicles' safety envelopes at any given moment. The abstract model assumes *perfect knowledge*, i.e., the sensors always provide the exact size of a safety envelope. All vehicles are able to access the infinite path *pth*, current segment index *curr*, and current position *pos* of any other vehicle through the current Traffic Snapshot. Using this information in conjunction with the size of a vehicle's safety envelope enables all vehicles to compute the shape of any vehicle's safety envelope represented by a set of segments and intervals describing the occupied space on each segment. However, their perception is limited by the view: A vehicle C can only perceive the parts of other vehicles' safety envelopes that are positioned on lane and crossing segments in its view $V(C)$. Note that parts of another vehicle's safety envelope may be visible even if its physical shape is not included in the view.

Claims and reservations can also be accessed through the Traffic Snapshot. Since claims are a virtual concept only used for coordination, the sets of claimed segments *cclm* are always accessible by all vehicles in the network, independent of their views. Reservations are, however, only perceived for the segments that are visibly occupied by the reserving vehicle's safety envelope. This causes reserved crossing segments $s \in \mathit{cres}(C)$ to be perceived as free as soon as C has physically passed the segments instead of only after C has withdrawn its reservation.

Analogously, this definition implies that reserved crossing segments *in front of* the reserving vehicle are also perceived as free if the safety envelope has not yet reached those segments. It must be mentioned here that although this behavior fits the formalization in the original publications [1], [4] and the intuitive meaning of reservations, it does not entirely reflect the authors' intention: Because reservations are a key component of the coordination protocol, perceiving a reserved crossing segment as free before it has been passed by the reserving vehicle can lead to unnecessarily dangerous situations. For example, two vehicles can reserve and approach the same crossing segment while being unaware of the conflict until the first vehicle's safety envelope reaches the segment. This should not be possible because both vehicles were aware of each other's intended maneuvers when their claims were placed; there is no reason to lose this knowledge when the claims are turned into reservations. A simple solution to this problem could be stretching the perceived safety envelope of vehicles that have reserved crossing segments so that it extends at least until the end of the crossing maneuver.[1] In the following, it is assumed that all reserved crossing segments that are within or in front of the reserving vehicle's safety envelope are perceived as reserved.

2.1.2 Urban Multi-Lane Spatial Logic

For reasoning about traffic situations in a road network of the abstract traffic model, the formal logic *Urban Multi-lane Spatial Logic (UMLSL)* is used. It is a purely *spatial* logic, which means that only positional and structural properties of traffic situations at *single points in time* are considered; the logic cannot express the order in which different situations occur or the duration over which a certain property holds. UMLSL formulas are evaluated over the views of individual vehicles in conjunction with the current Traffic Snapshot. Thereby, they can be used to reason about traffic situations from the perspective of a single vehicle rather than the whole traffic network.

Since UMLSL formulas only evaluate to either *true* or *false*, they are best used to check whether a specific property holds in the current view. For example, a UMLSL formula can be used by the ego vehicle to check if any other vehicle C that is visible in ego's view is currently driving on the same lane as ego. But finding out *which* vehicles satisfy this condition is not achievable by that formula alone; it can, however, be used as an effective tool to define the set of such vehicles easily: Let $sl(a,b)$ be a UMLSL formula that is *true* if and only if the vehicles identified by the variables a and b drive on the same lane, and $\mathbb{V}$ denote the set of all vehicles in the network. The set of vehicles driving on the same lane as ego can then be defined as $\{c \in \mathbb{V} \mid sl(\text{ego}, c)\}$, assuming that sl is evaluated over the view $V(\text{ego})$.

One of the intended applications of UMLSL formulas is the definition of vehicle controllers. While the *actions* that were mentioned above serve to describe the active behavior of a vehicle, the perceptual capabilities are modeled with the help of appropriate UMLSL formulas that are evaluated over the vehicle's view. This is demonstrated with the controller for safe intersection crossing maneuvers that is presented in Section 2.2.

[1]This discrepancy between the formalization and the intended behavior as well as the possible solution have been verified by Maike Schwammberger, the author of [1].

UMLSL can also be used to formally prove properties of the controllers using it in their definition. The safety of the controller for intersection maneuvers, for instance, is formalized and proven with the use of a *safety property* expressed in UMLSL. This formula, called *Safe*, is satisfied if and only if there exists no pair of vehicles whose safety envelopes overlap on any lane or crossing segment in the network. It suffices to evaluate the formula over the local views of all vehicles because it can be assumed that each vehicle perceives its own safety envelope and, thereby, any other vehicles' safety envelopes that might overlap with it. The proof itself begins with the assumption that the initial traffic situation satisfies *Safe*, and proceeds with showing that every possible transition, applied to a safe state, preserves the safety property. By the principle of induction, this proves that every thereby reachable traffic situation satisfies *Safe*. Using UMLSL both for the definition of the controller and the safety property makes the chain of arguments concise and easy to follow.

Before presenting how the vehicle controller works, the UMLSL formulas used for its definition should be explained. The controller requires only five relatively simple formulas; their intuition is easy enough to understand that no formal definitions are needed. Each of the formulas is evaluated over the view of the ego vehicle.

1. *col*(ego) (*Collision check*): This formula is a local version of the safety property *Safe* used for the safety proof. It is satisfied if ego's reservation overlaps with the reservation of any other vehicle.

2. *ca*(ego) (*Crossing ahead check*): The *crossing ahead check* is satisfied if and only if ego is driving on a lane segment, the distance between its reservation and the next intersection is less than a constant d_c, and there is no other vehicle in between. Intuitively, the ego vehicle does not need to care about crossing maneuver coordination as long as the formula is not satisfied. The constant d_c is set to a value that ensures sufficient space to react to an approaching intersection but avoids unnecessary actions while driving on a lane segment.

3. *pc*(c) (*Potential collision check*): If ego has an active claim, it can use the *potential collision check* to find out whether it is safe to keep the claim to turn it into a reservation later. The formula is satisfied if and only if ego's claim overlaps with the claim or the visible reservation of the vehicle c. In this case, ego should withdraw the claim to avoid a collision. This check is crucial for the controller's coordination procedure.

4. *lc*(ego) (*Lane change check*): The *lane change check* is satisfied while ego performs a lane change maneuver. Due to the way such maneuvers are executed in the abstract model, with ego holding reservations on two parallel lanes at the same time, no vehicle should commence a crossing maneuver during a lane change.

5. *oc*(ego) (*On crossing check*): This formula is satisfied if and only if any part of ego's reservation is situated on a crossing segment. It is useful to determine when a crossing maneuver starts or ends.

2.2 Crossing Controller

The abstract traffic model defines a vehicle controller to enable coordinated intersection crossing maneuvers. But this controller alone does not suffice to ensure a safe driving behavior of the vehicles under all circumstances. Therefore, the whole vehicle control system comprises two additional controllers: A *distance controller* that maintains a safety distance to vehicles in front and ensures that an intersection is not entered before the required segments are reserved, and a *lane change controller* that coordinates maneuvers on lane segments, such as lane changes, as long as no intersection is nearby. The latter is based on the controller for the country road scenario presented by Hilscher, Linker, and Olderog [18]. However, neither of the additional controllers is presented in detail because the crossing controller is the main focus; it is just assumed that the distance and lane change controllers work as intended and cause no unsafe situations. Unless stated otherwise, the term *(vehicle) controller* refers specifically to the crossing controller for the remainder of the thesis.

The coordination protocol is based on the same idea as the controller's predecessors for the motorway [3] and country road [18] scenarios. There, the goal is to perform a safe lane change maneuver by coordinating with nearby vehicles. The ego vehicle first places a claim on the neighboring lane segment it plans to drive on, which is similar to setting the turn indicator. The claim has the same extension as its safety envelope and can be used to check for potential collisions with other safety envelopes using an MLSL formula similar to the *potential collision check pc* of UMLSL. If there is no potential collision, the ego vehicle will turn its claim into a reservation and then physically perform the lane change maneuver. Otherwise, it will withdraw its claim and try the same approach again later.

The same basic procedure is used for the coordination of crossing maneuvers by the crossing controller. However, due to the vastly different network layout of an intersection compared to two parallel lanes, some adjustments are necessary. Most importantly, a vehicle must claim all required crossing segments at once to avoid deadlocks; the `cc` action is defined to meet this requirement. The analogy to a *deadlock*, a problem that is commonly found in concurrent programming scenarios, emphasizes the intuition of crossing segments serving as a shared resource with a mutual exclusion condition.

This is enforced by the definition of safety envelopes on crossing segments: As discussed in Section 2.1.1, reserved crossing segments are always occupied entirely, independent of the actual extension of the reserving vehicle's safety envelope. This causes any *potential collision check* performed by another claiming vehicle to detect a collision if one of the claimed segments is already reserved, regardless of the exact position of the reserving vehicle. The only exception to this are segments that have already been crossed; these segments are safe to be used by other vehicles even though they might still be marked as reserved in the current Traffic Snapshot.

For the formalization of the crossing controller, the concept of *timed automata* [19] is extended to incorporate UMLSL and the defined actions that vehicles can perform to change the traffic situation. Timed automata are an extension of the common *finite automata* used to describe the behavior of systems with time constraints. The main contribution of this concept are *clock variables* that can be used by an automaton to monitor and react to the passing of time. A clock variable x is a real-valued variable that can be used by an automaton in two ways: When used in a *transition guard*, the variable can be part of an expression like $x < 3$ that specifies a condition that must be satisfied for the transition to be available. As part of a *transition action*, clock variables can be reset to zero ($x := 0$). This is the only way the value of a clock variable can be changed actively. When a specific amount t of time elapses, all clock variables of the automaton are increased by t.

The abstract traffic model extends this concept to define *Automotive-Controlling Timed Automata (ACTA)*. Transition guards of an ACTA are extended to allow Boolean expressions of clock constraints and UMLSL formulas, while the transition actions additionally allow for the claim and reservation actions defined in section 2.1, now called *controller actions*. The new syntax of transition labels is

$$\varphi / a; c, \tag{2.1}$$

where φ is the transition guard, $a \in \{\texttt{cc}, \texttt{rc}, \texttt{wd\_cc}, \texttt{wd\_rc}\}$ is the controller action, and c is a list of clock variable resets of the form $x := 0$. Additionally, each state of an ACTA has a *state invariant* expressing a condition that must be satisfied while the state is active. Transition guards and state invariants can have the same type of expressions. State invariants can therefore block or force transitions based on clock variables and UMLSL formulas.

The formal definition of the crossing controller as an ACTA is depicted in Figure 2.2. It uses one clock variable x and has five states $q_0, \ldots, q_4$, each with a clear intuition of the current situation of the controlled vehicle ego provided by the state invariants. Each vehicle must start in a safe situation, which is expressed by the state invariant $\neg col(\text{ego})$ of the initial state q_0. As soon as ego approaches an intersection, the formula $ca(\text{ego})$ is satisfied and the transition to q_1 is available. The states q_1, q_2, and q_3 all specify this as part of their invariant, meaning that the vehicle must stay in front of the intersection as long as one of these states is active. The next transition to q_2 has no guard and performs the `cc` action. Hence, the vehicle will place its claim on the required crossing segments immediately after detecting the intersection.

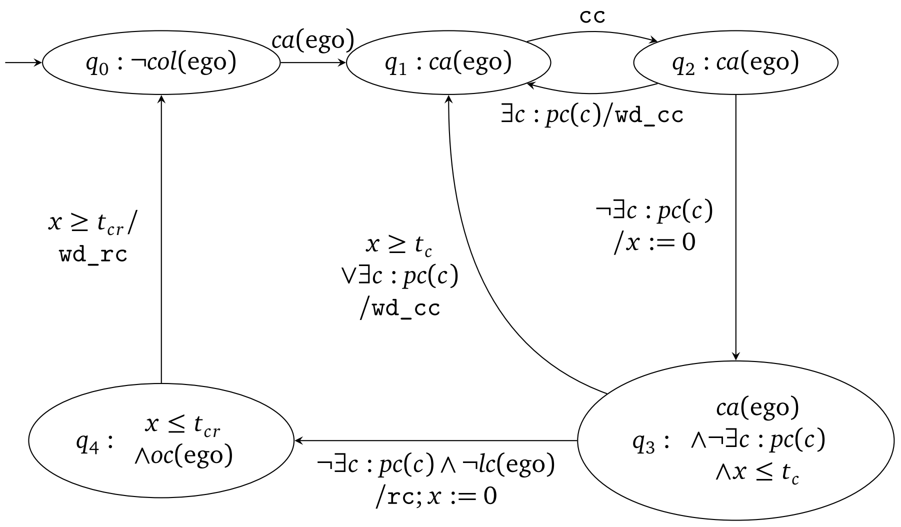

$q_0 : \neg col(\text{ego})$
$ca(\text{ego})$
$q_1 : ca(\text{ego})$
cc
$q_2 : ca(\text{ego})$
$\exists c : pc(c)/$wd_cc
$x \geq t_{cr}/$ wd_rc
$x \geq t_c \vee \exists c : pc(c)$ /wd_cc
$\neg\exists c : pc(c)$ $/x := 0$
$q_4 : x \leq t_{cr} \wedge oc(\text{ego})$
$\neg\exists c : pc(c) \wedge \neg lc(\text{ego})$ /rc; $x := 0$
$q_3 : ca(\text{ego}) \wedge \neg\exists c : pc(c) \wedge x \leq t_c$

Figure 2.2 – ACTA definition of the crossing controller used in the abstract traffic model. This is a slightly simplified copy of the automaton shown in [1, Figure 8].

In the next state q_2, both outgoing transitions have the formula $\exists c : pc(c)$ in their guard, but one is negated. This means that after the claim was placed, the vehicle will perform a *potential collision check* for every visible vehicle and initiate a transition to either q_1 or q_3 based on the result. If a potential collision is detected, the transition to q_1 will be taken and the `wd_cc` action will withdraw the vehicle's claim; the same process will repeat itself until no potential collision is found. Once this is the case, the controller transitions to q_3 and resets the clock variable x, which can be interpreted as starting a timer. In state q_3, the formula $\exists c : pc(c)$ is, again, used in the guards of all outgoing transitions and additionally in the state invariant. This models the vehicle repeatedly verifying that its claim does not overlap with any other vehicle's claim or reservation. If a potential collision is detected in this state, the vehicle will withdraw its claim and return to q_1, just as before in q_2. The same transition is also triggered if the clock variable x exceeds the predefined constant t_c. This is a time threshold limiting the duration a claim can be active before it must be withdrawn or turned into a reservation. It is implemented in the controller as a precaution to avoid deadlocks.

The transition from q_3 to q_4 is only available if the claim is still safe and ego is not performing a lane change maneuver. If the vehicle is not changing lanes or manages to finish the maneuver in time, it may perform the transition and reserve the claimed crossing segments, initiating the crossing maneuver. This will also reset the clock variable to start a timer again. State q_4 is active while the vehicle performs the crossing maneuver and some part of its reservation occupies a crossing segment, as indicated by the UMLSL formula $oc(\text{ego})$ in the invariant. As soon as the maneuver is finished or the timer exceeds the time threshold t_{cr}, the controller makes a transition back to the initial state q_0, withdrawing the reservation. The constant t_{cr} is assumed to be the longest time a crossing maneuver performed by any vehicle on any intersection in the network can take. Just like t_c, it is a precaution to ensure that the crossing reservation is withdrawn properly and timely.

Note that this controller does not provide any information on more detailed vehicle controls, such as throttle and brake levels or steering angle. This task is delegated to the mentioned distance controller and additional control systems on a lower abstraction level.

2.3 Simulation Framework

The simulation part of this thesis (Chapters 4,5) employs the Open Source vehicular network simulation framework *Veins* [6]. Veins couples the discrete event simulation engine OMNeT++ [20] with the microscopic traffic simulator SUMO [21] and provides a basis for the development of applications for intelligent vehicles as well as simulations for evaluating their performance. This thesis uses Veins version 5.0, running in OMNeT++ 5.5.1 and with SUMO 1.2.0.

From a high-level perspective, OMNeT++ models a communication network as a set of *modules* that can exchange messages among each other. The process of sending and receiving a message can be customized to implement arbitrary, highly detailed communication models. For example, it is possible to model perfect communication, where every sent message is received and processed instantly, as well as highly realistic wireless communication taking into account physical effects of radio transmission, antenna properties, environmental conditions, and other factors that can influence communication delay and reception probability.

Veins incorporates facilities that leverage this functionality to implement realistic wireless communication models at the level of the physical layer of an IVC protocol stack. By representing every vehicle in a SUMO traffic simulation as a module in a parallel OMNeT++ network simulation, it enables the vehicles to communicate with each other. Developers can write programs at the application layer to control the communication as well as the driving behavior of each individual vehicle. This bidirectional coupling of the two simulations is the main feature of Veins and the foundation for implementing the crossing controller from the abstract traffic model.

Before moving on to the implementation in Chapter 3, however, it is important to discuss the basics of SUMO's road network and traffic representation in more detail and compare them to the abstract model.

2.3.1 Traffic Representation in SUMO

SUMO (Simulation of Urban MObility) is a microscopic traffic simulator developed by the Institute of Transportation Systems at the German Aerospace Center. It can be used to accurately model road networks, define highly detailed traffic and vehicle characteristics, and run city-scale simulations with a plethora of parameters and performance metrics.

Road networks in SUMO are represented by directed graphs. In contrast to the abstract traffic model, however, SUMO uses the edges to describe roads, and intersections are represented by single nodes. Both nodes and edges have more detailed information attached to them; directed edges, for example, represent a set of individual, parallel lanes that lead in the same direction, and each of them has its own set of properties describing the physical shape, maximum speed, supported vehicle types, and more.

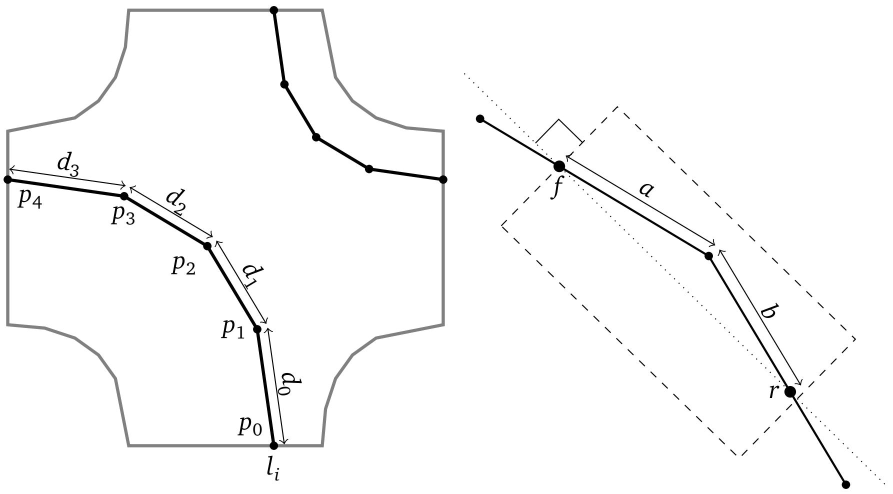
Nodes that represent intersections have even more complex information on the intersection's layout and right of way rules. The shape of an intersection is described by a polygon in the two-dimensional plane that represents the network's ground area. Every physical object in the simulation has a position and shape in this plane. Possible turning maneuvers of intersections are represented by so-called *connections*. A connection stores the identifiers of one incoming and one outgoing lane as well as the turning direction, which can be straight, left, right, or U-turn.

Compared to the abstract model's intersection representation, the existence of a connection from lane l_0 to l_1 in SUMO can be interpreted as the existence of a directed path of crossing segments between the two lane segments that represent l_0 and l_1 in the abstract network graph. For example, the left-turning maneuver from lane 1 to lane 6 in Figure 2.1a could be represented by a connection with the content $(1, 6, \text{"left"})$ in SUMO, while the abstract network graph shown in Figure 2.1b allows this maneuver implicitly through the existence of the directed path $(l_1, c_1, c_2, c_3, l_6)$. Observe that the abstract model has no notion of a turning direction and generally allows more turning maneuvers. The directed paths between incoming and outgoing lane segments are not necessarily unique and the implicit definition of permitted maneuvers is less restrictive than SUMO's explicit method; the network graph in Figure 2.1b would require a significantly different structure if the intersection did not allow U-turn maneuvers, while in SUMO removing the respective connections would suffice.

To specify the exact trajectories of vehicles performing a crossing maneuver, SUMO assigns an *internal lane* to each connection. Internal lanes are structurally identical to the regular lanes between intersections. Each lane in the network has a unique identifier and a *shape*. The shape of a lane l is a list of points in the plane $P_l = \{p_0, \dots, p_n\} \subset \mathbb{R}^2$ that describe the lane's center line. This list is always ordered in driving direction and it starts and ends with the exact start and end points p_0, p_n of the lane. The length $\text{len}(l)$ of a lane is defined as the total length of all line segments between two consecutive points p_i, p_{i+1}, which can be computed as the Euclidean distance $d(p_i, p_{i+1})$:

$$\text{len}(l) := \sum_{i=0}^{n-1} d(p_i, p_{i+1}) \tag{2.2}$$

Therefore, every real value in $[0, \text{len}(l)]$ identifies a position on the lane l that is situated on one of the line segments. Figure 2.3a shows the layout of a small intersection and two of its internal lanes.



(a) Intersection shape with internal lanes **(b)** Schematic vehicle shape computation

Figure 2.3 – Intersection shape and computation of vehicle shapes. Figure 2.3a shows the polygon describing the shape of an intersection in gray and two internal lanes with their center line points. The internal lane l_i has length $\text{len}(l_i) = d_0 + \ldots + d_3$. In Figure 2.3b, the orientation calculation of a vehicle is illustrated. The real-valued position p of the vehicle identifies the point f. Using the vehicle's length l, the point r at position $p - l$ is obtained by moving backwards along the line segments $(a + b = l)$. The dotted line runs through f and r and is used to position the vehicle's final shape, represented by the dashed rectangle.

These positions are used to describe the current locations of vehicles: Every vehicle in the network has a current lane and a real-valued position on that lane describing the center point of its front bumper. Note that the abstract model describes positions in a similar way but uses the rear bumper instead. The shape of a vehicle is a rectangle whose length and width are specified by the vehicle's *type*. Thus, all vehicles of the same type have the same shape, but there can be multiple vehicle types of varying shapes.

Given the length and position of a vehicle, its orientation in the network is computed as follows: First, the length is subtracted from the position of the front bumper, yielding another position that is possibly on one of the previous line segments or even another lane. The angle of the line through the two positions is calculated and used as the orientation angle of the vehicle. The final vehicle position is then obtained by placing the rectangle that describes the shape so that it fits the vehicle's front bumper position and the orientation angle. A schematic view of this process is depicted in Figure 2.3b.

The purpose of SUMO's internal lanes is solely the spatial description of the trajectories that are followed by the vehicles performing the respective crossing maneuvers. In the abstract traffic model, the trajectory of a crossing maneuver is described by a sequence of crossing segments. Because the abstract model is inherently less specific about vehicle positions, the crossing segments do not provide the same level of spatial information as an internal lane describing the same maneuver. However, crossing segments additionally define all possible crossing maneuvers as well as their incoming and outgoing lanes through their directed edges, i.e., they incorporate the information that SUMO represents separately as connections. On top of that, they constitute the shared resource that is essential for the coordination protocol. Although at a first glance, SUMO's internal lanes and the abstract model's crossing segments appear to be similar concepts, their applications are vastly different and they reveal one of the most significant differences between the abstract traffic model and SUMO.

Apart from the structure of the road network, SUMO offers various ways of specifying traffic characteristics. A straightforward method that is also very similar to the abstract model is defining the route of each vehicle individually. A route is usually specified as a list of contiguous edges, much like the infinite sequence of lane and crossing segments that specifies a path in the abstract traffic model. Unlike the infinite sequences, however, routes in SUMO are usually finite and will cause a vehicle to spawn at a certain time, follow its route, and then exit the simulation. If only the start position and target edge of a route are provided, the vehicle will generate its route based on the road network and various routing parameters. In general, vehicles can intelligently determine their own routes, find the correct lanes that are required for their crossing maneuvers, and reroute dynamically to avoid traffic jams. SUMO also provides suitable *car-following models* for controlling the vehicles' behavior on a smaller scale as well. They can be used to ensure realistic acceleration and deceleration rates, safe lane change maneuvers, and safety distances, which fits the purpose of the abstract model's distance and lane change controllers perfectly.

Chapter 3

Controller Implementation

The evaluation of the vehicle controller's performance in a more realistic environment requires a suitable implementation in the simulation framework. This chapter presents the development of such an implementation.

Running traffic simulations to investigate the controller's performance in the presence of various possibly deteriorating effects is the main purpose of the developed implementation. However, the implementation process itself and the entailing design considerations may provide valuable insights into the advantages and disadvantages of the abstract approach to controller specification.

To provide a holistic view of this process, I will first address general, theoretical design considerations (Section 3.1), before moving on to more detailed descriptions of the individual design problems and the implemented solutions. The main challenges to be discussed are bringing together the abstract model's and SUMO's traffic and coordination concepts (Section 3.2), and the technical realization of the vehicle control system (Section 3.3). In Section 3.4, I will present a number of practical problems that occur when using the implementation without additional adjustments, enabling a first assessment of the abstract model's limitations. Finally, I will report on the verification and validation methods used to ensure that the developed implementation functions appropriately in Section 3.5.

3.1 Controller Design Considerations

Capturing the intended behavior of the vehicle controller as precisely as possible is crucial for the acquisition of valid and meaningful simulation data. Therefore, this should be the primary goal of the implementation.

The definition of the controller comprises two components: The automaton describing its behavioral aspects, and the abstract traffic model with UMLSL describing its environment and how it is perceived. The automaton can be seen as an active component that requests and processes traffic information from the passive abstract model, using UMLSL as a query language. It is an obvious and natural design decision to implement the controller automaton as part of a vehicle application such that every vehicle is controlled by an individual instance. In order to provide access to the required traffic information, it is possible to create a separate module within Veins that acts as an interface between all controller applications and the simulation framework. This architecture provides full control over the way the traffic information is represented and accessed, and enables the vehicles to communicate by using Veins' communication models.

Thereby, both components of the abstract controller definition have suitable representations. Each of them can be adjusted wherever it is necessary to achieve a viable system complexity and performance, as long as the final result is sufficiently similar to the original.

Because the abstract model's purpose is not only the abstract representation of arbitrary urban traffic situations but also the formalization and proof of properties possessed by vehicle controllers defined within the model, many of its features are irrelevant for the controller's behavior and can, therefore, be omitted. The traffic model and UMLSL are means to represent and reason about vehicle controllers on a high level of abstraction. This can be leveraged by replacing the abstract concepts with more convenient and application-specific components, which is one of the main benefits of using abstraction in general.

In this case, the abstract concepts that carry the most potential for simplification, but also require the most care when being replaced, are the abstract model's intersection representation and UMLSL as a whole.

The way traffic networks with multi-lane roads and, more importantly, intersections are represented in the abstract model is designed to be both flexible enough to support a great variety of network structures, and simple enough to avoid unnecessary complexity caused by non-essential information. This suggests that the traffic model is well suited to be used for the implementation without significant changes. Additionally, the traffic model employed by SUMO is relatively similar in terms of network structure and vehicle representation, and already provides most of the required functionality. For example, both models represent traffic networks using a directed graph structure with additional information on the road length and number of lanes, but SUMO also provides facilities for network generation and routing. Such functionalities are not required by the abstract model and would need to be implemented manually, introducing unnecessary complexity and potential for errors. Instead, simple translations between the two representations are sufficient to represent many aspects of SUMO's traffic model in a way that the vehicle controller can understand, while retaining the convenience functionalities provided by SUMO.

However, this is only possible for the general road structure and vehicle positions outside of intersections. The translation of one intersection representation into the other must be implemented with great care, since intersections show the most significant differences between the models and play a role of critical importance for vehicle coordination. This matter is discussed in more detail in Section 3.2.1.

The idea of replacing components from the abstract model with facilities provided by the simulation framework can be applied to UMLSL as well, albeit in a somewhat different manner. The abstract controller definition uses UMLSL formulas for transition guards and state invariants. However, UMLSL has a complete formal definition and is supposed to be used for the description of arbitrary vehicle controllers and proof-carrying as well. This functionality is entirely irrelevant for the controller that is to be implemented since only a very small number of formulas are actually used in its definition (cf. Figure 2.2). These formulas have relatively simple and intuitive semantics and therefore lend themselves to be implemented directly in dedicated functions.

Considering that, for a more general solution, a kind of formula evaluation system for UMLSL would be required, only the above alternative seems feasible. It is worth mentioning, however, that some progress has been made in this area of research: It was shown by Fränzle, Hansen, and Ody [22] that a subset of MLSL, dealing with a bounded number of vehicles, is decidable. The presented methodology has, to my best knowledge, not been extended to UMLSL so far. Although the algorithm was not designed to be suitable for use in a real-time application, this approach might provide a viable alternative to direct formula implementations in the future.

This decision is supported by the simulation framework since, as mentioned above, it provides programmatic access to all required information through SUMO. In addition to this, implementing UMLSL formulas directly allows the view construction to be adjusted so that only information relevant to a specific formula is computed. This also alleviates the task of adapting the intersection model since customizing the view construction removes some requirements, yielding higher flexibility. Furthermore, implementing only what is necessary for the controller to work reduces complexity and improves efficiency, which is beneficial in the simulation context and, presumably, in a real-world scenario too.

From a more technical perspective, the implementation should satisfy several requirements to ensure good usability regarding simulation studies. For example, defining a variety of input parameters and output metrics should be a straight-forward modification, and the system should be transparent enough that any observed effects can be traced back to their causes easily. Although the simulation framework already provides most of this functionality, keeping the implementation structured and simple can help with the elimination of errors and the general understanding of the system. A high computational complexity should be avoided because it can limit the scope and granularity of parameter studies. However, correct controller behavior and sufficient clarity have a higher priority and should not be sacrificed for efficiency.

In conclusion, the general implementation guidelines that result from the major theoretical design considerations can be stated as follows:

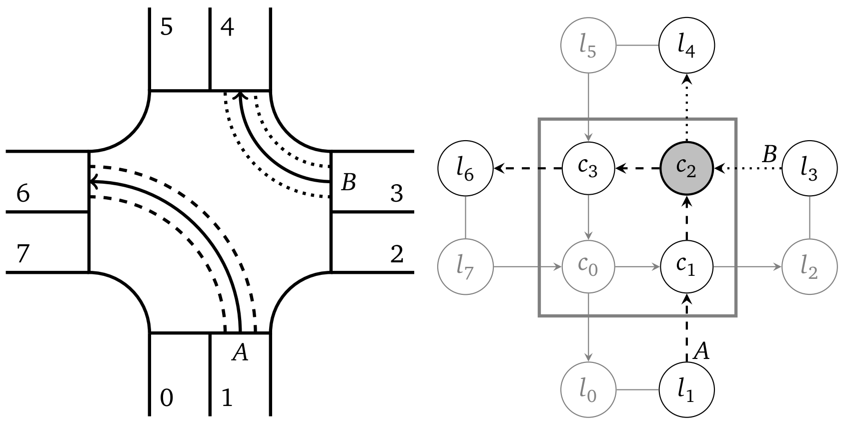
- The implementation architecture comprises a vehicle application containing the controller automaton and a separate component in Veins acting as an interface between the controller and the traffic model.
- Abstract concepts that are not required by the controller definition in their entirety should be simplified and represented by facilities of the simulation environment, or omitted completely, if possible.
- None of the adjustments should significantly alter the behavior of the vehicle controller unless the original behavior is unreasonable within the more realistic simulation scenario.
- The desired result is an implementation that exhibits the same behavior in the simulation environment as the original controller shows in its abstract environment. It should provide the means to define and run simulation studies with reasonably low effort.

The following sections deal with the main implementation challenges in more detail. Note that programming-related issues are not discussed and all addressed challenges and solutions are presented independently of any programming language.

3.2 Traffic Model Interface

Because the vehicle application relies on the functionality provided by the traffic model interface and provides a greater design flexibility, the traffic interface should be developed first. Its main purpose is providing all traffic information required by the controller automaton in the form of queries defined by UMLSL formulas. In order to implement these formulas adequately, a suitable interpretation of traffic information from the simulation framework and particularly SUMO is required.

As described in Chapter 2, the intersection structure employed by SUMO displays the most significant difference to the abstract traffic model, while the roads outside of intersections are represented in very similar ways. It is not obvious how an intersection described in one of the models can be translated into the other without losing or altering – or having to devise – too much information. In order to represent a SUMO intersection in terms of the abstract model, one would have to define a set of crossing segments with directed edges such that every possible maneuver has a corresponding directed path of the correct length and each pair of conflicting maneuvers shares at least one segment. It would also be desirable for maneuvers to share segments *only* if they are in conflict to avoid unnecessary waiting times. Figure 3.1 illustrates this problem for a small example.



(a) Detailed maneuver trajectories **(b)** Segment paths of maneuvers

Figure 3.1 – Comparison of maneuver representations. Figure 3.1a shows a left turn maneuver of a vehicle A and a right turn of vehicle B on the 2×2 intersection from Figure 2.1. Figure 3.1b shows the corresponding segment sequences in the abstract network graph. The maneuvers share crossing segment c_2 although the real trajectories are not in conflict. It is unclear what size each crossing segment should have and how the graph should be changed to model the real situation more accurately.

This applies to the vehicles crossing the intersection as well: The description of a maneuver depends on the representation of the underlying road structure. Consequently, a maneuver represented within one model of the road structure may have no or multiple equivalent representations in the other model. This is especially problematic because a proper representation of intersection crossing maneuvers is crucial for the coordination protocol to operate correctly.

However, it is not necessary to represent an intersection and its crossing maneuvers in exact accordance with the abstract model in order to create sensible implementations of the controller queries. A closer inspection of the crossing segments and their uses leads to an alternative that combines SUMO's detailed trajectory description and the coordination features required by the crossing controller.

3.2.1 Intersection Model Adaptation

The most visible function of the crossing segments is defining the layout and area of an intersection. The connections between the segments and the road network clearly define the incoming and outgoing lanes, and the directed edges between the segments provide information on their placement within the intersection. As mentioned previously, the segment shapes are not defined precisely. But the number of segments and their sizes can be used to get a rough estimation of the size and shape of an intersection and the crossing maneuvers.

The possible turning directions and shapes of the maneuvers are also defined by the directed edges. However, their definition is rather implicit and their correct usage depends entirely on how the vehicle paths are defined. In addition to that, the paths do not describe the exact trajectories followed by the vehicles, but rather possible connections between incoming and outgoing lanes. For example, the abstract model can describe a regular 2×2 intersection and a roundabout connecting the same roads with the same graph structure [1, Figure 4], although the real trajectories of the vehicles are very different.

Finally, the most important purpose of the crossing segments for the vehicle controller is serving as a shared resource for coordination. By enforcing mutual exclusion on the segment reservations of the vehicles, the controller ensures the safety of all maneuvers. Therefore, it is crucial to construct a suitable representation of this functionality in the new model.

In short, the crossing segments serve three main purposes:

1. Describe the high-level *layout* of the intersection.
2. Define *connections* between incoming and outgoing lanes and the *segments required by each maneuver*.
3. Serve as a *shared resource* for coordination.

These purposes are only loosely related to each other; the intersection layout gives no indication of permitted maneuvers, the segment paths defining the maneuvers exist on a much lower level than the rough intersection layout, and the coordination functionality does not depend on either of the two whatsoever. Observe that both the intersection layout (1) and the possible crossing maneuvers (2) are not required by the controller: The layout is not used at all apart from the coarse route definitions of the vehicles. Because the routes are predefined and the controller cannot change them, there is no need to couple this information so tightly with the crossing maneuvers. The same applies to the definition of possible maneuvers by connections of incoming and outgoing lanes. Since all of this information is already directly accessible through SUMO, it is not necessary to consider these aspects for the new intersection model design.

The coordination functionality of the crossing segments (3) is the only aspect that has no direct counterpart in SUMO. Therefore, it is necessary to design a new system based on the available information. It is helpful to make several observations on the abstract model that can simplify this problem: First, every intersection has a relatively small number of possible crossing maneuvers. It is reasonable to assume that every vehicle that performs a specific maneuver chooses the same trajectory and, in terms of the abstract model, the same path of crossing segments. Thus, each possible maneuver has a uniquely defined set of crossing segments that can be determined statically. Because of this, it suffices to have the controller consider only *sets of segments* instead of individual segments for coordination.

Secondly, there is no need to check whether two maneuvers intersect at run time: Given two sets of crossing segments, they either have at least one segment in common or they are disjoint. Using the first observation, this major part of the coordination protocol can also be determined statically. Transferring this concept to SUMO, the information whether two maneuvers, or internal lanes, conflict with each other is static and can be computed as such. As the collision check used for the coordination protocol is mainly based on checking whether a set of claimed crossing segments intersects with another set of segments, this information is useful for the vehicle controller. More importantly, however, it can be computed for internal lanes *without the use of segments* by using the lanes' physical shapes instead. This result can be considered to be a solution to the problem of matching crossing segments with internal lanes that was discussed previously.

Before moving on to details on how this can be calculated, a critical question must be answered: Is this approach valid, i.e., does it produce behavior equivalent to the abstract model? To find the answer, consider an arbitrary intersection with two vehicles, each trying to perform a crossing maneuver. If the internal lanes describing the trajectories do not overlap, a collision will be impossible and the vehicles can perform their maneuvers without any coordination. When using the new intersection model, the claims and reservations of the vehicles do not overlap since these conflicts are computed directly using the lanes' shapes. In the abstract model, it is not guaranteed that the two sets of crossing segments describing the maneuvers do not intersect. However, both cases are acceptable: If the sets are disjoint, the vehicles will perform their maneuvers without being disturbed, as before. If the sets do intersect, the vehicles will have to coordinate and cross one after the other; in this case, the abstract model is just unnecessarily restrictive. It is reasonable to assume that this is an effect of applying the abstract model to a realistic intersection and does not reflect the intended purpose of the segments.

If, on the other hand, the trajectories overlap, a collision will be possible and the vehicles must not cross the intersection simultaneously. By definition, the new model detects this conflict and causes the vehicles to behave appropriately. If the sets of segments from the abstract model intersect, the vehicles will do the same. If for some reason, the sets are disjoint, the vehicles will be unable to actively avoid a collision even though the abstract safety property is not violated.

This shows that, in terms of crossing maneuver safety, the new intersection model is equivalent to the abstract model. However, due to the lost maneuver granularity that was provided by the crossing segments, this model alters the vehicle behavior: Since vehicles only place claims and reservations on whole internal lanes instead of sets of multiple segments, reservations stay active for the whole duration of the maneuver. In the abstract model, a vehicle's position, as perceived in the view of another vehicle, is used to determine whether a reserved segment is free again after the reserving vehicle has passed it. By implementing the view interpretation and evaluation of UMLSL formulas appropriately, this behavior can be restored to the new model. As mentioned in Section 3.1, UMLSL formulas will be implemented directly and require no explicit view model. Therefore, a similar, static solution is possible by combining the trajectory shape and implicit view information: Given two internal lanes l_1, l_2 whose shapes intersect and a vehicle driving on l_2, there exists a position p on l_2 at which the vehicle leaves the conflict zone of the two lanes. The part of the trajectory that remains when the vehicle reaches p does not intersect with l_1 any more. Another vehicle can safely claim and reserve lane l_1 once the first vehicle has reached p. Since these positions can be computed statically, just like conflicts of lane shapes, and vehicles can use their perception of other vehicles' positions to determine when it is safe to place a claim again, the new intersection model can accommodate this functionality.

With this problem solved as well, the new intersection model is complete. Its major design aspects and functionalities can be summarized as follows:

- The **intersection structure of SUMO** is used directly. Internal lanes and their physical shapes serve as coarse representations of the vehicles' real trajectories. What is defined by directed edges between segments in the abstract model is represented by connections between incoming and outgoing lanes in the new model.

- **Claims and reservations** are placed on internal lanes instead of crossing segments. Each crossing maneuver can be represented by a fixed set of segments; two maneuvers conflict with each other if their respective sets intersect. The same inquiry amounts to intersecting the shapes of internal lanes in the new model. In both cases, the information is static and can be computed as such.

- **Gradual reservation clearance** is enabled by providing the exit positions of conflict zones. For each pair of intersecting internal lanes, the two lane positions at which the zone of conflict with the respective other lane is passed are computed statically.

3.2.2 Static Data Structure Calculation

To provide more details on how the new intersection model is implemented, the techniques used to calculate and store the static information are explained in the following. The running example for this section is a simple 2 × 2 intersection as computed by SUMO. Its shape and internal lanes are depicted in Figure 3.2. On each of the four incoming lanes, it is permitted to drive straight, turn left, or turn right. This leads to a total of 12 internal lanes which are numbered 0 . . . 11; the numbering starts with lanes coming from the North, moving clockwise around the intersection, and numbering first right-turning lanes, then straight, and then left-turning lanes for each direction. The width of the internal lanes is set to $w = 2.5\,\mathrm{m}$. For simplicity, each lane only has five points describing its center line. SUMO automatically uses only two points to describe the straight internal lanes.

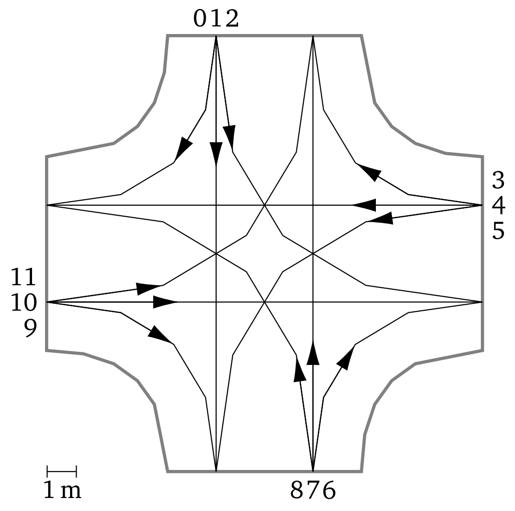


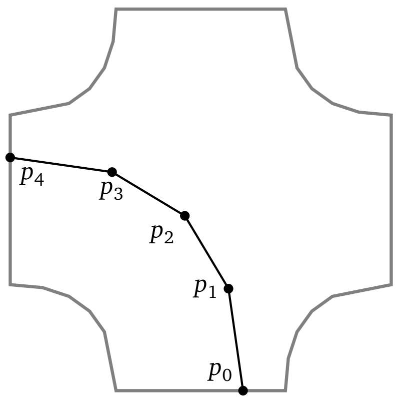
Figure 3.2 – Example 2 × 2 intersection from SUMO with 12 internal lanes.

To start with, the shapes of all internal lanes must be intersected to determine which maneuvers are in conflict. Due to the structure of this information, it is stored in a matrix-like data structure that is henceforth called the *foe matrix* F. The foe matrix needs to store one Boolean value for each pair of internal lanes. Therefore, it is a matrix of the format $F \in \{0, 1\}^{12\times 12}$. Because it is symmetrical, only half of its values actually need to be computed and stored; for simplicity, it is treated as a full matrix here.

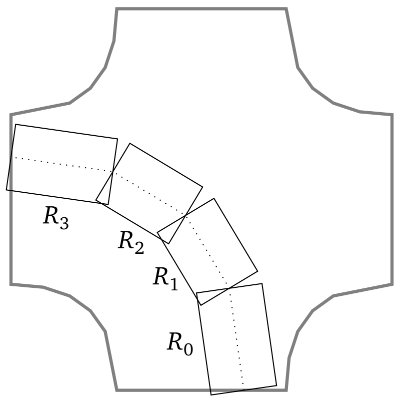
It should be mentioned that the employed version of SUMO also computes foe information for every intersection to realize its built-in management policies. However, this is based on heuristics for right of way rules and generally produces different results than intersecting the internal lanes.

To compute the entries of the foe matrix, the shapes of all internal lanes are used. For these computations, the lane shapes are approximated by widening the center lines to the specified width w. This ensures that possible maneuver conflicts can be detected even if the center lines do not intersect, given that w does not under-approximate the real maneuver shapes. Additionally, the lane shapes can be reused to compute conflict zone entry and exit positions later.

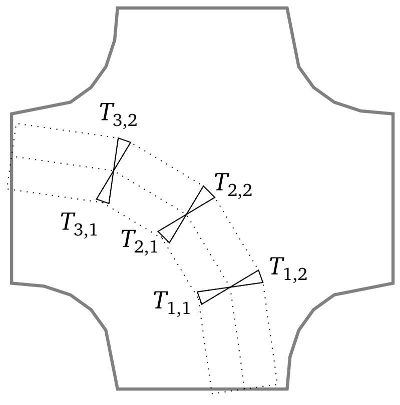
Let $l \in \{0, \dots, 11\}$ be an internal lane and $P_l = \{p_0, \dots, p_n\} \subset \mathbb{R}^2$ the list of points describing its center line. P_l contains two or five points in this scenario ($n \in \{1, 4\}$), depending on whether the internal lane l is a straight or a turning lane. For each pair of consecutive points p_i, p_{i+1}, we construct a rectangle R_i such that one side has a length of w and the other side is parallel to the line that connects p_i and p_{i+1}, as depicted in Figure 3.3b. This yields n rectangles $R_0, \dots, R_{n-1}$, where R_i is based on the line connecting p_i and p_{i+1}. For $n > 1$, it is likely that consecutive rectangles are positioned at an angle to each other, leading to gaps between them. These gaps can be filled using triangles $T_{i,1}, T_{i,2}$, one on each side of the point p_i that R_{i-1} and R_i have in common. This step is illustrated in Figure 3.3c.



(a) Center line points



(b) Rectangles added



(c) Triangles added to fill gaps

Figure 3.3 – The process of computing the approximate shapes of internal lanes as sets of polygons for the left-turning lane with index 8.

Finally, the shape of the internal lane is approximated by the set of rectangles and triangles

$$S(l) := \{R_0, \dots, R_{n-1}\} \cup \{T_{1,j}, \dots, T_{n-1,j} \mid j \in \{1, 2\}\}. \tag{3.1}$$

These simple polygons can be computed efficiently and suffice for approximating the crossing maneuver's required area, especially when using a larger number of points for the center line. Additionally, they enable the application of an efficient algorithm for polygon intersection which is necessary for the foe matrix computation.

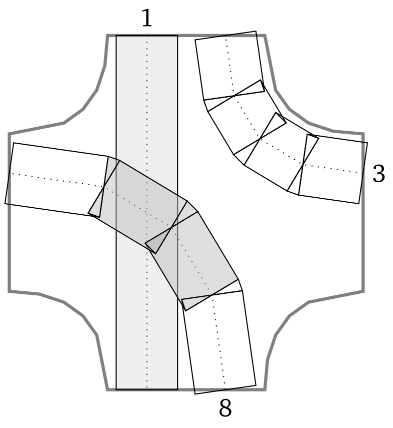
Let Intersect$(p, q) \in \{0, 1\}$ denote the output of a Boolean function that determines whether polygons p and q as constructed above, intersect. Having constructed the set of polygons $S(l)$ for each internal lane l, determining whether two approximate shapes overlap is as simple as intersecting each pair of polygons from their respective sets: Let $l_1, l_2 \in \{0, \dots, 11\}$ be two internal lanes, then

$$l_1 \text{ overlaps with } l_2 :\iff \exists p \in S(l_1), q \in S(l_2) : \text{Intersect}(p, q) = 1. \tag{3.2}$$

The foe matrix can be computed using this information as shown in Algorithm 3.1. To illustrate the intuition, Figure 3.4 depicts the relation between overlapping shapes of internal lanes and entries of the foe matrix using a small example.

```
Input: Set of internal lanes $L = \{l_0, \dots, l_{n-1}\}$ with shape approximations
Output: Foe matrix $F \in \{0,1\}^{n \times n}$ based on overlapping lanes
1: $F \leftarrow n \times n$ matrix filled with 0s
2: for $i = 0, \dots, n-1$ do
3:     $F_{i,i} \leftarrow 1$  // Lane overlaps with itself
4:     for $j = i+1, \dots, n-1$ do
5:         $x \leftarrow \begin{cases} 1 & \text{if } l_i \text{ and } l_j \text{ overlap} \\ 0 & \text{otherwise} \end{cases}$
6:         $F_{i,j} \leftarrow x$
7:         $F_{j,i} \leftarrow x$  // Matrix is symmetrical
8:     end for
9: end for
10: return $F$
```

Algorithm 3.1 – Foe matrix computation using approximate shapes of internal lanes.



(a) Overlapping shapes of internal lanes

$$\begin{array}{c|ccccccc|}
 & \cdots & 1 & \cdots & 3 & \cdots & 8 & \cdots \\
\hline
\vdots & & & & & & & \\
1 & & 1 & & 0 & & 1 & \\
\vdots & & & & & & & \\
3 & & 0 & & 1 & & 0 & \\
\vdots & & & & & & & \\
8 & & 1 & & 0 & & 1 & \\
\vdots & & & & & & &
\end{array}$$

(b) Resulting entries in foe matrix

Figure 3.4 – Example of foe matrix entries for three internal lanes. Lanes 1 and 8 overlap while lane 3 has no conflict with either of the two. Note that each lane obviously also overlaps with itself.

The exit positions of conflict zones to be used for gradual reservation clearance (in the following called *clearance positions*) are computed and stored in a similar way. For each pair of overlapping internal lanes (l_i, l_j), the clearance position marks the point that a vehicle traveling on l_j must have reached so that another vehicle may safely claim and reserve l_i. This shortens the waiting time of vehicles and mimics the effect of reserved crossing segments in the abstract model that become free to claim and reserve as soon as the reserving vehicle has passed them.

To compute these positions, the shapes of the internal lanes can be used again. However, due to the way vehicles are positioned on internal lanes in SUMO, the polygons and center line points of a lane shape are not accurate enough to determine the clearance position on the lane itself. Instead, the rectangular shape of a vehicle performing a crossing maneuver is simulated and intersected with the approximate shape of the foe lane. Given the internal lanes (l_i, l_j), the vehicle positions on l_j can be iterated in small steps and in reverse order until the vehicle shape overlaps with the approximate shape of l_i. The previous position is then stored in another matrix $C \in \mathbb{R}^{n \times n}$, where n is the number of internal lanes, at position (i, j). In general, it is possible that the first vehicle placement already intersects with the foe lane. In this case, the vehicle is moved forward instead and the clearance position is the first where no overlap is determined. This position is located on the outgoing lane of the maneuver. Figure 3.5 illustrates this computation for a single pair of internal lanes where the vehicle is moved backwards with a relatively large step size.

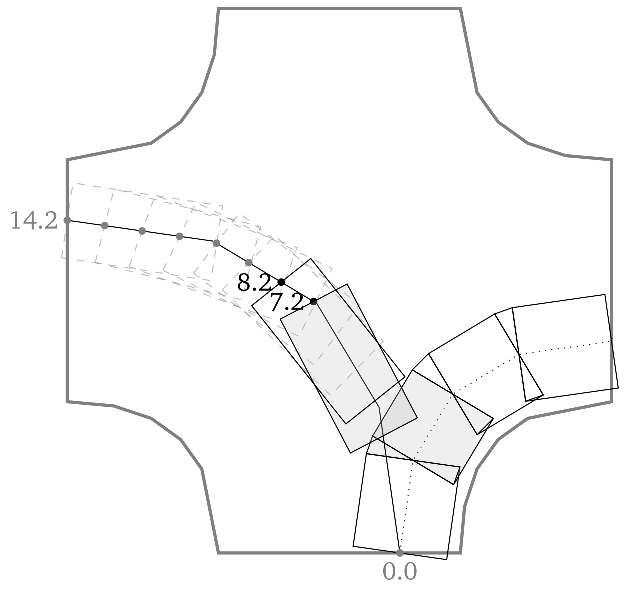


Figure 3.5 – Computation of the reservation clearance position for the internal lanes 6 and 8. The length of lane 8 is 14.2 m and the step size is 1 m. The vehicle's rectangular shape is 2 m wide and 4 m long. Position 7.2 is the first position at which the vehicle's shape overlaps with the approximate shape of lane 6. Therefore, the previous position is the resulting clearance position: $C_{6,8} = 8.2$. As soon as a vehicle on lane 8 has reached this position, it does not prevent other vehicles from using lane 6 any more.

3.3 Vehicle Controller Application

Building on the newly constructed intersection model, this section discusses the implementation of the vehicle controller as a Veins application. To this end, I will first present the way vehicles perceive and represent their environment in comparison to the universally accessible data used in the abstract model. This includes the introduction of a minimal communication protocol to transfer the coordination mechanics from the global Traffic Snapshots to purely local vehicle interactions. Afterwards, I will outline how the controller automaton is realized in software and how the necessary UMLSL formulas are integrated. Finally, I will explain how the application exerts control over its vehicle based on the controller actions.

In the abstract traffic model, vehicles get information on the traffic situation from two different sources: The current Traffic Snapshot and their local view. The Traffic Snapshot contains the relative lane positions, claims and reservations of all vehicles while the view of a vehicle provides more detailed information on the vehicles that are perceived by the vehicle's sensors, such as their safety envelope. Both are used to evaluate the UMLSL formulas that serve as transition guards of the vehicle controller. Therefore, vehicles must have access to all traffic information the controller requires.

However, the Traffic Snapshots in the abstract model are virtual, globally synchronized data structures that are accessible by all vehicles. Such data structures do not exist in a realistic scenario and, thus, they should not exist in the simulation as well. Instead, each vehicle has an internal traffic model that contains all necessary information and is kept up to date by the vehicle itself. It is reasonable to assume that vehicles are equipped with on-board sensors that are able to gather all necessary physical information. To model this in the controller application, it suffices to query the simulation framework. If inaccuracies and errors in the sensor data are of interest, they can be applied to the real data easily. However, claims and reservations are virtual properties that must be acquired by other means. Since the vehicles are able to communicate via wireless channels, it is obvious to utilize this ability for coordination by synchronizing the vehicles' internal information on claims and reservations. Natural opportunities to send synchronization messages are the controller actions that change claims or reservations. Because the vehicles will only need to coordinate for crossing maneuvers, the relevant controller actions are `cc`, `wd_cc`, `rc`, and `wd_rc`. Since the new intersection model uses internal lanes instead of crossing segments as coordination resource, claims and reservations only have to apply to single internal lanes. Using this information, the messages that represent the four controller actions can all have a similar structure and carry a small amount of data:

- The message type
- The identifier of the sending vehicle
- The identifier of the affected internal lane (only `cc` and `rc`)

All vehicles will send the corresponding message when one of these controller actions is performed. When a vehicle receives a message of this type, it can update its internal traffic model accordingly. These are the only messages required for coordination.

To keep the semantics of controller actions as clear as possible, the vehicles apply the changes implied by a received message immediately and without any validation. For example, when a vehicle A receives a message of the type `rc` for lane l from vehicle B, A must update its internal model to reflect that B now holds a reservation on l and does not have a claim any more (since reservation actions remove claims automatically), regardless of whether B held a claim or reservation on l or any other lane before. Because all vehicles are equipped with the same controller and follow the same coordination protocol in the simulation scenario, it is safe to rely on this simple model.

The next task is the realization of the vehicle controller automaton. As described in Section 2.2, the abstract specification uses an extended timed automaton, or Automotive-Controlling Timed Automaton (ACTA), to define the behavioral aspect of the controller. It is a straightforward task to implement the basic structure of a finite automaton in software; evaluating the transition guards and performing the transition actions is, however, more involved. Before these operations are implemented, it should be established *when* they are executed, i.e., when the automaton is triggered. The semantics of automata demand that transitions are made as soon as a transition guard evaluates to *true*. State invariants affect this behavior by forcing (blocking) a transition when the invariant of the current (next) state is *false*. These conditions can be interpreted as extensions of the regular transition guards and are therefore treated similarly.

In the simulation environment, time passes in discrete but irregular time steps and the state of the traffic situation over which transition guards are evaluated can change with each step. By evaluating the transition guards in each time step, the automaton could function with the highest accuracy possible. However, this would tie the vehicle behavior to the inner workings of the simulation framework and, most likely, cause many unnecessary evaluations because not all simulation events affect the relevant vehicle properties. Furthermore, it would imply that the vehicles' sensors and on-board computers can react with arbitrary precision and latency, which is highly unrealistic. Instead, the vehicle controller automaton is triggered at fixed, regular intervals that can be configured to be small enough that the vehicles retain a sufficient and realistic reaction time without causing performance issues. Each time the automaton is triggered, the guards of possible transitions are evaluated and the corresponding transition and action are performed based on the result.

The only exception to the regular executions are violated time constraints: Resetting a clock variable schedules a simulation event at the time this variable will exceed its threshold. If the variable is relevant to the automaton's current state when the event occurs, the automaton is triggered immediately.

Having specified *when* the automaton is triggered, it is time to define more precisely *what happens* at each trigger event. There are only four relevant controller actions (`cc`, `wd_cc`, `rc`, and `wd_rc`), and their definition is straightforward: Each action places or withdraws either a claim or a reservation on an internal lane. The executing vehicle therefore modifies its internal traffic model correspondingly and broadcasts the respective message, causing the same modification to the internal models of all receiving vehicles. Only the `rc` action has an additional effect as it also removes all claims of the reserving vehicle due to the action's semantics. These actions are executed every time the respective transition is made.

Deciding when a transition occurs requires the transition guards and state invariants to be evaluated. Since all of them are defined by UMLSL formulas and clock variables, it suffices to implement these constituents and then construct the final formulas using logical operators. As already discussed above, clock variables are represented by special timer events. The controller automaton uses only one clock

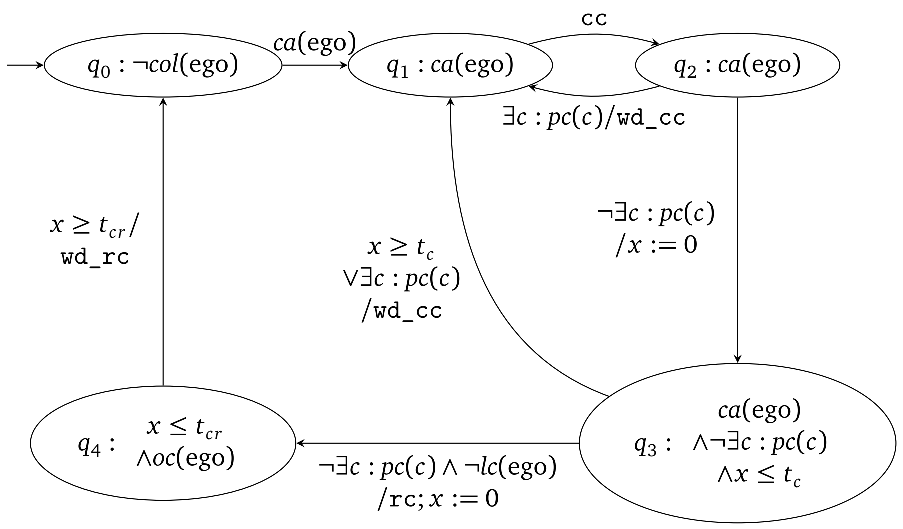

$q_0 : \neg col(\text{ego})$
$ca(\text{ego})$
$q_1 : ca(\text{ego})$
cc
$q_2 : ca(\text{ego})$
$\exists c : pc(c)/$wd_cc
$\neg\exists c : pc(c)$ $/x := 0$
$x \geq t_{cr}/$ wd_rc
$x \geq t_c$ $\vee\exists c : pc(c)$ /wd_cc
$q_4 :$ $x \leq t_{cr}$ $\wedge oc(\text{ego})$
$\neg\exists c : pc(c) \wedge \neg lc(\text{ego})$ /rc; $x := 0$
$q_3 :$ $ca(\text{ego})$ $\wedge\neg\exists c : pc(c)$ $\wedge x \leq t_c$

Figure 3.6 – Crossing controller ACTA definition (Copy of Figure 2.2 for ease of reading).

variable x, but two different thresholds t_c and t_{cr}. Each time the timer is reset, timer events for both thresholds are scheduled in the simulation framework. The timer variable will then be smaller than the threshold until the respective event occurs; for the controller trigger caused by the event, it will be equal to the threshold, and afterwards, it will be larger until the timer is reset again. This leaves only the UMLSL formulas to be implemented. In the following, the implementation of each of the five formulas used by the automaton is presented.

(1) $col(\text{ego})$**:** This formula is only used by the controller as state invariant for q_0. Its main purpose is ensuring that each vehicle starts in a safe situation, which can be assumed to be satisfied by the simulation environment since SUMO will only spawn vehicles in positions where immediate collisions can be ruled out. Apart from that, the formula is irrelevant for the transition from q_4 to q_0 because the transition action `wd_rc` does not affect the vehicle's driving beyond the intersection and, if the state invariant does not hold, the controller has already failed its purpose and a transition to q_0 cannot change that. Therefore, it is not necessary to implement this formula at all.

(2) $ca(\text{ego})$**:** This formula checks whether the vehicle is currently approaching an intersection, the distance is smaller than some constant d_c, and there is no other vehicle in between. It is used as transition guard for $q_0 \rightarrow q_1$ and as state invariant in all states that are responsible for the coordination procedure (q_1, q_2, and q_3). It is reasonable to assume that a vehicle that intends to cross an intersection and to follow the coordination protocol will not turn away from the intersection or change lanes once it starts with coordination. Therefore, the only relevant application of the formula is guarding the transition that initiates the coordination. The identifier i and length l of the current lane, the vehicle's position p, and its braking distance d_{brake} can be acquired from the simulation framework. Suitable representations of these properties are provided by the traffic interface. It is also used to check that no other vehicle is driving on the same lane between the ego vehicle and the intersection. The formula then only evaluates to *true* if i meets an intersection and $l - p + d_{brake} < d_c$ holds, which is trivial to implement.

(3) $pc(c)$**:** The check for potential collisions is the main component of the coordination protocol. All states that can be active while the vehicle has placed its claim (q_2 and q_3) use it to decide whether it is safe to continue or the claim must be withdrawn. Since all occurrences of the formula are bound by an existential quantifier $\exists c : pc(c)$, it can be integrated into the implementation. As discussed in Section 3.2.1, all vehicles have access to the foe matrix F and the reservation clearance positions C through the traffic model interface. Additionally, a vehicle can perceive the positions of the relevant other vehicles with its sensors; again, the traffic interface provides a suitable representation of this information. In conjunction with the vehicle's internal traffic model, the collision check can be performed as shown in Algorithm 3.2.

Input: Claimed lane l, foe matrix F, clearance positions C, and ego's internal claim and reservation information
Output: *true* if ego's claim overlaps with another vehicle's claim or reservation, *false* otherwise

```
1:  for each internal lane $c$ claimed by another vehicle do
2:    if $F_{l,c} = 1$ then  // $l$ and $c$ are foes
3:      return true  // Collision with other claim
4:    end if
5:  end for
6:  for each internal lane $r$ reserved by another vehicle $B$ do
7:    if $F_{l,r} = 1$ then  // $l$ and $r$ are foes
8:      $p \leftarrow$ position of $B$  // Maybe $B$ is far enough
9:      if $p < C_{l,r}$ then
10:       return true  // Not far enough: collision with $B$'s reservation
11:     end if
12:   end if
13: end for
14: return false  // No collision found
```

Algorithm 3.2 – Implementation of *potential collision check* $\exists c : pc(c)$.

(4) *lc*(ego)**:** This formula checks whether the ego vehicle is currently performing a lane change maneuver and is used to ensure that an intersection is not entered while changing lanes. Since in the employed lane change model of SUMO, lane changes happen instantaneously, this check is not necessary. Apart from that, vehicles should not change lanes after placing a claim anyway because this would completely change their trajectory and invalidate the coordination procedure.

(5) *oc*(ego)**:** The *on crossing check* holds as long as the ego vehicle is driving on the intersection, i.e., occupying a part of an internal lane with its safety envelope. It is used by the automaton as state invariant of q_4 and helps detecting when the reservation can be withdrawn. Similar to the *crossing ahead check ca*(ego), only sensory information is required to evaluate this formula. The traffic interface provides all information that is required to track the extension of the ego car's safety envelope and check if any part of it is located on the intersection.

All of the formula implementations, timer events and automaton trigger mechanisms described above are used to realize the vehicle controller application. Every vehicle spawned into the simulation has its individual traffic model and controller with the automaton starting in the initial state q_0.

There is one more aspect of the controller application that needs to be discussed. So far, the only effects of the controller on the vehicle are sending coordination messages and updating the internal traffic model. Without additional changes, the vehicles will follow SUMO's default driving behavior and intersection policy since the implementation lacks a way for the controller application to exert control over the vehicle. In the abstract model, the intersection controller is accompanied by a distance controller maintaining safety distances and safe speeds, and a lane change controller that employs a reservation-based coordination protocol for safe lane change maneuvers. It is not necessary to implement these controllers manually because SUMO already provides systems to perform these tasks. To make the vehicles behave as the intersection controller demands, the existing control systems can be reconfigured so that only a few direct steering commands are required.

First, the vehicle controls can be adjusted to ignore intersection policies like traffic lights or right of way rules. This causes vehicles to maintain a safe speed during crossing maneuvers and the safety distance to other vehicles on the same lane but perform crossing maneuvers as if no other vehicles were using the intersection. The lane change behavior is unaffected by this adjustment.

Next, the vehicles need to stop and wait at an intersection until the controller has managed to place a reservation on the required internal lane. SUMO provides a *stop* command that can be used to make a vehicle stop at a certain position for a specific duration. It is intended to be used for bus stops and taxis, or to artificially create traffic jams, and causes a smooth, realistic deceleration similar to a vehicle slowing down to stop at a red light. The *stop* command is used by the vehicle controller application as follows: Each vehicle that enters a lane meeting an intersection receives the command to stop at the end of this lane for an unlimited duration. As soon as the controller reserves the required internal lane, i.e., the automaton enters state q_4, the stop is canceled and the vehicle proceeds to cross the intersection.

In addition to waiting at intersections, vehicles must be prevented from changing lanes after selecting an internal lane for their crossing maneuver. This can be enforced by another small adjustment that reconfigures the lane change controller to stay on the current lane as soon as the first claim is placed.

These features suffice to make the vehicles behave in accordance with the controller application. However, this basic implementation has a number of inherent problems that cause unintended behavior and necessitate further adjustments in order to make the system viable for simulation studies.

3.4 Practical Implementation Problems

The controller implementation developed in the previous sections aims to capture the intended behavior of the original abstract controller as precisely as possible. However, when used in simulations for validation, it displays some unexpected phenomena that may have various causes; most likely, the abstraction level introducing some freedom for interpretation is among the main reasons. Since these problems occur frequently, impede the simulation process, and are, presumably, not part of the intended behavior, they must be examined and resolved. They also serve as basis for an evaluation of the disadvantages imposed by the highly abstract controller definition, as they can indicate design problems that must be considered for a real implementation. This topic is discussed in Chapter 5.

The most apparent problem manifests itself in a total blockade of an intersection due to a livelock of two or more vehicles trying to claim their respective lanes. To illustrate the emergence of such a situation, consider two vehicles A and B approaching an intersection from different directions and having conflicting maneuvers. According to the controller automaton, a claim is placed with the transition from q_1 to q_2, and the check for potential collisions using the formula $pc(c)$ is performed only in the subsequent states. If a potential collision is detected, the claim will be withdrawn; otherwise, it is kept and turned into a reservation after another collision check in state q_3. Combined with the fact that due to the same trigger frequency, the vehicles' controllers are always triggered in the same order, this behavior leads to an infinite loop of both vehicles placing and withdrawing claims, as illustrated in Figure 3.7.

	State		Claim placed		
t	A	B	A	B	Action
0	q_1	q_1	-	-	
1	q_2	q_1	X	-	A places claim
2	q_2	q_2	X	X	B places claim
3	q_1	q_2	-	X	A checks for collision - withdraws claim
4	q_1	q_3	-	X	B checks for collision - keeps claim
5	q_2	q_3	X	X	A places claim
6	q_2	q_1	X	-	B checks for collision - withdraws claim
7	q_3	q_1	X	-	A checks for collision - keeps claim
8	q_3	q_2	X	X	B places claim
9	q_1	q_2	-	X	A checks for collision - withdraws claim
⋮	⋮		⋮		Infinite loop

Figure 3.7 – Livelock caused by two vehicles A, B with conflicting maneuvers. Both vehicles start in state q_1 and approach the intersection at the same time from different directions. Their controllers are always triggered in the same order. The state sequence from time steps 3 to 8 will repeat itself indefinitely as long as no exterior influence disturbs the system.

Analyzing this problem, the main cause becomes fairly obvious: The coordination technique uses internal lanes as a shared resource with a partial mutual exclusion condition. If two lanes overlap, only one of them may be reserved at a time. By claiming the required lane and checking for a potential collision, a vehicle can check whether the lane is currently in use. However, while the claim is active, other vehicles can detect it as a potential collision although the vehicle that placed it is also just checking for collisions. The reason for this unintuitive behavior is that by placing a claim, a vehicle already uses the lane, and that a whole trigger interval passes before it performs the collision check. From another perspective, this resembles a well known problem in computer science; multiple agents access a shared resource concurrently and the operation that tries to acquire the resource is *not atomic*. To fix this, the operation of claiming a lane, checking for potential collisions, and then withdrawing or keeping the claim must be made atomic. This can be realized by forcing the automaton to leave state q_2 immediately, i.e., within the same simulation time step. By doing so, at the end of the time step, the vehicle will either remain in q_1 without a claim or have an active claim in q_3 without any potential collisions. To avoid unnecessary messages, sending the `cc` message can be moved to the transition from q_2 to q_3, and the `wd_cc` message at the transition from q_2 to q_1 can be removed completely. This effectively makes the claim and potential collision check atomic and thereby solves the livelock problem.

Note that this adjustment lies completely within the scope of free design decisions because the abstract model does not define an execution model for the controller automaton. It does not violate any theoretical obligations or assumptions and externally displays a behavior that is consistent with its internal actions due to the adjusted messages. Therefore, it is not detrimental to the validity of the implementation.

The next problem that occurs when using the controller implementation without further changes is related to the *oc*(ego) formula and the way vehicles approach intersections. Examining the automaton transition from state q_3 to q_4 closely, a peculiar timing behavior becomes obvious: The formula *oc*(ego) is part of the state invariant of q_4 and must therefore hold immediately after the transition actions are performed. One of these actions is `rc`, i.e., placing the reservation. However, the formal definition of *oc*(ego) demands that a part of the ego vehicle's safety envelope is already present on the intersection. In the abstract model, this is not a problem because vehicles are technically allowed to start their maneuver before the reservation is placed. But the way SUMO's *stop* command is used to make vehicles wait in front of an intersection as long as no reservation is placed makes this impossible. Therefore, no vehicle is able to perform the transition and start its crossing maneuver.

Simply ignoring the state invariant for the transition does not solve the problem: If a vehicle places its reservation long before entering the intersection, the formula will detect that it is not driving on the intersection in the next step, since it has not even entered it yet, and force a transition to q_0, withdrawing the reservation prematurely. This behavior can, however, be prevented by modifying the implementation of *oc*(ego) so that it also evaluates to *true* if the vehicle is *in front of* the intersection and has placed its reservation.

This adjustment causes reservations to be placed earlier. However, by enabling this behavior, vehicles can approach intersections without having to slow down if they are able to place a reservation quickly enough. It also should not change the order in which vehicles perform their maneuvers because the time at which claims are placed is unaffected and the `rc` action only becomes relevant when a claim is already established. The only noticeable implication is that the crossing maneuvers can last slightly longer due to the added travel distance; this must be taken into account when selecting a value for the crossing time threshold t_{cr}.

The next two problems occur when introducing simulation parameters for unreliable communication. Only the loss of messages and communication delay are of interest here. The complete list of simulation parameters is presented in Chapter 4.

A major feature of the controller and traffic model implementation is that vehicles have to synchronize their internal traffic information by exchanging messages. Because the consequences of lost or delayed information are of interest for evaluating the controller's performance, these effects were introduced artificially by implementing suitable simulation parameters. However, when verifying their correct functionality in test simulations, they frequently caused intersections to be blocked, similar to the livelock problem discussed above. Since this behavior makes reasonable simulation studies impossible, further adjustments to resolve these issues were necessary. Again, the implications of these problems are evaluated in Chapter 5.

When a *message delay* is introduced, a certain time passes before received messages are processed. This can lead to asynchronous internal traffic models, causing decisions of the vehicle controllers based on outdated information. For example, it is possible that two vehicles try to place their claims on conflicting lanes at almost the same time, so that neither of the claim messages is processed before the other message is sent. The result is that both vehicles will have to withdraw their claims the next time their controllers are triggered. Afterwards, it is likely that the same process happens again since the controllers have the same trigger interval.

A related problem occurs when *message loss* is introduced, i.e., some messages are not received at all. Losing messages of the type `cc` or `rc` can lead to collisions: The potential collision check of a vehicle that misses some claim or reservation information may find no collision and cause the vehicle to reserve a lane that overlaps with another reserved lane. However, losing a `wd_cc` or `wd_rc` message can be much more problematic in the simulation scenario because it can prevent vehicles from ever reserving their required lane. This applies especially to `wd_rc` messages since they are, in general, only sent once per intersection.

Both of these problems can be characterized as faults of the communication protocol. It is likely that a more sophisticated method with techniques for detecting message loss and ensuring a generally more reliable exchange of information can solve them. However, designing a reliable communication protocol is not a goal of this thesis; investigating the effects of unreliable communication on the controller's performance, on the other hand, is. Therefore, the solution to these communication issues should only serve to allow for stable and reliable *simulations* without blocked intersections. It should not affect the ability to systematically introduce communication errors through simulation parameters.

The implemented solutions to both mentioned problems utilize time stamps. To start with, messages of the type `cc` are extended by a time stamp field storing the time at which the sender placed the claim. Because it is highly unlikely that two vehicles with conflicting maneuvers place their claims at the exact same time, this information can be used to prioritize the claim of the vehicle that placed it first. The reaction to receiving a claim message is adjusted accordingly: If the vehicle has no active claim or the claimed internal lanes do not overlap, it will only update its internal traffic information as before. However, if a potential collision is detected, there are two possible cases: (1) the other vehicle's claim is older than the receiving vehicle's, or (2) it is newer.

In case (1), the claim of the receiving vehicle has lower priority and must be withdrawn immediately. This is achieved by updating the internal traffic model with the new claim and causing an immediate trigger of the controller automaton.

In case the other claim is newer (2), the receiving vehicle has higher priority and the other vehicle should withdraw its claim. In order to keep the communication protocol simple, the claim with lower priority is just ignored in this case. Since all vehicles are equipped with the same controller, the other vehicle can be assumed to behave accordingly.

The second problem mentioned above arises when messages are lost and the internal traffic information of a vehicle does not reflect the real situation. Since only the case where `wd_xx` messages are lost and claim or reservation information is left in the internal traffic model is detrimental to the simulation, the solution focuses on this problem. As described in Chapter 2, the abstract traffic model specifies two time thresholds t_c and t_{cr} for the time that passes between placing a claim and turning it into a reservation, and the time between placing the reservation and withdrawing it, respectively. Consequently, a claim can never be active longer than t_c and no reservation can be active longer than t_{cr}. It is therefore safe to delete information on claims and reservations that is older than these time thresholds. This is accomplished by storing claims and reservations in the internal traffic model together with the time the message containing the information was processed. Every time this data is accessed, the expired entries are removed.

With these fixes in place, the controller implementation is complete. It should be mentioned here that all of the adjustments discussed in this section only marginally alter the behavior of the vehicles and serve to improve the simulation quality and reliability. Given that the abstract model cannot provide solutions to all of these minor problems due to its high level of abstraction, some small changes to the exact behavior that leave the overall intuition of the coordination protocol intact are acceptable.

3.5 Verification and Validation

Due to the lack of a reference implementation and execution model for the abstract traffic model and crossing controller, validating the implemented application is particularly challenging. Additionally, the controller implementation and adapted intersection model do not portray the abstract traffic model directly, but rather a concrete instantiation of its abstract concepts in a more realistic environment. This is a structural difference to usual validation scenarios because the original system does not provide an exact specification of the features the developed model and implementation require; the modeled system is usually less abstract than the implementation to validate. For instance, the abstract traffic model has no mechanism to assert liveness, i.e., ensuring that every vehicle eventually reaches its destination. It also does not specify a communication model, speed limits, or concrete values for any of the declared constants. All of these aspects need to be considered by the implementation to ensure sensible simulations with a high degree of realism.

This discrepancy between the original model and the developed system makes a traditional validation based on experimentation data infeasible. Instead, the controller validation is conducted on the basis of behavioral predictions and manual observation.

The verification, on the other hand, is more straightforward, and the implemented architecture can be divided into three components whose correct functionality can be verified individually: The computation of the foe matrix and clearance positions, as discussed in Section 3.2.2, the crossing controller automaton (cf. Section 3.3), and its integration into the vehicle control system that is responsible for sending coordination messages and controlling the vehicle.

To start with, the calculation of the foe matrix and clearance positions lends itself to graphical verification. Since all involved calculations deal with simple geometrical shapes in a two-dimensional plane, their correctness is easy to confirm visually by generating appropriate result illustrations comparable to Figures 3.4a and 3.5. Due to the relatively small size of the data structures, verifying them manually for each intersection layout that was selected for simulations was a simple task.

The implementation of the crossing controller automaton can be verified directly by tracing its states, transitions, and actions through a number of scenarios with predefined sequences of events for which the controller's behavior can be predicted using the automaton's formal semantics. Because the automaton only has a total of five states, a relatively small number of scenarios suffices to cover all possible transitions and state configurations. By decoupling the automaton from the vehicle control system, this process was carried out independently of the traffic simulation environment, removing any external effects it might introduce.

A similar approach was followed to verify that the rest of the vehicle control system that complements the automaton works correctly. This includes managing the data structures that represent the internal traffic model and evaluating the UMLSL formulas correctly, executing controller actions, reacting to received messages and timer events, and triggering the controller at the correct intervals. Most of these aspects can be verified independently of the traffic simulation just like the automaton using standard software development techniques like unit tests. To test the vehicle control functionalities, however, it was helpful to run minimal simulations and record vehicle traces and communication traffic, and observe the behavior in SUMO's graphical interface. By testing the controller application in such simulations with various ordinary and degenerate parameter settings continuously during the implementation process, the correct functionality of all its components is assured.

While the verification process asserts that the implemented system works as expected, appropriate validation is required to ensure that it correctly reflects the vehicle behavior and, more specifically, the coordination protocol defined by the abstract traffic model. As outlined above, however, validating this consistency is only possible to a certain extent since the simulation environment is more specific than the abstract model and not all of its features have suitable counterparts for validation. Therefore, the validation process is based heavily on running simulations and manually checking whether the observed behavior matches the abstract model's specifications.

For instance, a vehicle that approaches a clear intersection is expected to visit each state of the automaton exactly once and place or withdraw its claim and reservation accordingly:

$$q_0 \rightarrow q_1 \overset{\texttt{cc}}{\rightarrow} \underbrace{q_2 \rightarrow q_3}_{\text{Claim active}} \overset{\texttt{rc}}{\rightarrow} \underbrace{q_4}_{\text{Reservation active}} \overset{\texttt{wd\_rc}}{\rightarrow} q_0 \tag{3.3}$$

This simple case can be validated by recording states, transitions, and actions in a basic simulation scenario with one vehicle and one intersection. If the recorded state sequence is different, there must be an error in the controller application or the simulation scenario. A simulation like this revealed the problem with the *on crossing check oc* that was discussed in Section 3.4: The vehicle would not enter state q_4 nor start its crossing maneuver although it had already entered q_3; it would only return to q_2 after the time threshold t_c was exceeded, and then repeat this process. The described solution fixed this behavior, validating that the controller works correctly in the most basic case.

Generalizing this experiment to many vehicles approaching the same intersection on conflicting paths, only a few of them should be able to execute their crossing maneuver immediately and follow the state sequence 3.3. Instead, most vehicles should repeatedly place and withdraw their claims due to failing *potential collision checks*:

$$q_0 \rightarrow q_1 \overset{\texttt{cc}}{\rightarrow} q_2 \underbrace{\left([\rightarrow q_3] \overset{\texttt{wd\_cc}}{\rightarrow} q_1 \overset{\texttt{cc}}{\rightarrow} q_2\right)^*}_{\text{As long as } \exists c{:}pc(c) \text{ is } \textit{true}} \rightarrow q_3 \overset{\texttt{rc}}{\rightarrow} q_4 \overset{\texttt{wd\_rc}}{\rightarrow} q_0 \tag{3.4}$$

Recording the state sequence of each simulated vehicle in a structured format enables a simple, automatic way to ensure that this pattern is never violated without the need for manual observation. It should be mentioned here that the adjustment solving the livelock problem that was discussed in Section 3.4 does not affect this sequence for individual vehicles; it only affects how the state sequences of multiple vehicles intertwine (cf. Figure 3.7). As a matter of fact, the livelock problem was first detected using this technique. Moreover, this example shows why the validation of the controller application with respect to the abstract traffic model is problematic: The behavior that causes a livelock and thereby blocks the intersection does not technically violate the abstract model's specifications. It is rather an artifact of implementing an execution model that triggers the controller automata of the vehicles at regular intervals in combination with the non-atomic operation of placing a claim and performing a *potential collision check*.

Another aspect that is crucial for the coordination protocol and closely related to the sequence of states and actions is the management of claims and reservations. A straightforward approach to validating the correct management is tracing the current claims and reservations of all vehicles through a simulation and checking whether the traces match the occurrences of the controller actions and their semantics. However, to ensure that the coordination protocol is followed by all vehicles, the mutual exclusion condition of reservations must be considered as well. The formal *safety property* of the abstract model requires that no space on a crossing or lane segment is reserved by more than one vehicle at the same time. Because the car-following model of SUMO can be expected to take care of the safety on all lane segments, considering only crossing segments is sufficient. With the new intersection model presented in Section 3.2.1 in place, the safety property can be expressed in terms of internal lanes and clearance positions:

$$\begin{aligned} \mathit{Safe}^* \equiv \nexists c_i, c_j : \exists l_i \in \mathit{cres}(c_i), l_j \in \mathit{cres}(c_j) : & \qquad (3.5) \\ F_{i,j} = 1 \land \mathit{pos}(c_j) < C_{i,j} \land \mathit{pos}(c_i) < C_{j,i} & \qquad (3.6) \end{aligned}$$

The equation defines the adapted safety property *Safe** to hold if and only if there exist no two vehicles c_i, c_j with respective reserved lanes l_i, l_j (3.5) such that l_i and l_j overlap and both vehicles have not yet reached their respective clearance positions (3.6). This property can be checked automatically and must hold at all times and in all scenarios without additional parameters that introduce more realistic effects. Running simulations in the least realistic configurations showed that with all the presented fixes in place, the crossing controller implementation works as intended. Having validated that the implemented system satisfies the safety property for this basic case, it can be used as a baseline for further simulation studies.

Chapter 4

Simulation Setup

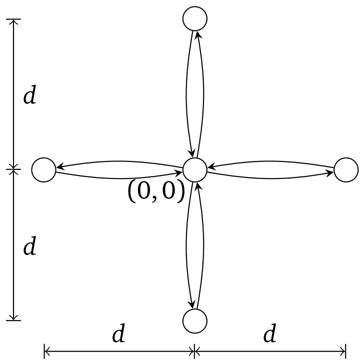
The simulative approach to evaluating the performance of the crossing controller offers a plethora of possible traffic scenarios, simulation parameters, and performance metrics to investigate. Because the abstract traffic model promises safety in arbitrary intersection scenarios, a great variety of different layouts and traffic configurations should be covered. As the main subject of this thesis is the safety performance of the controller in a realistic environment where many assumptions of perfect information and communication do not hold, a suitable selection of parameters and metrics must be defined to enable a thorough evaluation.

In this chapter, I present the setup of the simulation studies that were conducted for this performance evaluation. Section 4.1 deals with the definition of intersection scenarios and introduces a simple notation for the identification of the employed layouts. Building on this, the process of generating traffic is described in Section 4.2, along with details of the individual vehicle characteristics. Finally, Sections 4.3 and 4.4 define the simulation parameters and performance metrics, focusing on the introduction of imperfections encountered in a realistic environment and the assessment of the controller's safety properties.

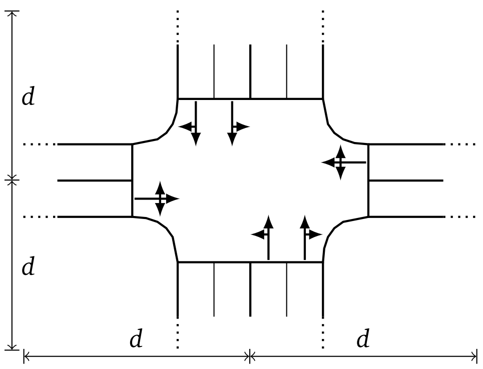
4.1 Intersection Scenarios

Each simulation scenario comprises one intersection with limited sections of its incoming and outgoing lanes. This simplifies the analysis of simulation results and provides full control over the intersection's structure and the incoming traffic. The main design aspects of intersections that are considered here are the number of intersecting roads, the number of their lanes, and the presence of dedicated turning lanes.

SUMO provides tooling for generating road networks from simple descriptions of nodes and edges. Individual edges and intersections can be defined with additional information such as the number of lanes and the connections describing the possible crossing maneuvers. The basic structure of every scenario is defined by a node at position $(0,0)$ representing the intersection under consideration and four additional nodes with coordinates $(\pm d,0)$ and $(0,\pm d)$, where d defines the distance of the outer nodes from the intersection node. The outer nodes are connected to the intersection node by edges in both directions, creating a network with four roads that meet in the middle and resemble a four-way intersection with 90° angles. Each of the four roads has a length of approximately d and a configurable number of lanes. Given just this information, SUMO computes the intersection shape with all internal lanes as well as the lanes of the incoming and outgoing edges.



(a) Simple network description



(b) Detailed intersection in SUMO network

Figure 4.1 – Intersection definition and generation with SUMO for a 2×4 scenario. Figure 4.1a shows the five nodes at their respective positions in the plane. Figure 4.1b shows the generated intersection and roads in the SUMO network. The road lengths are measured from the center of the intersection, so they are slightly smaller than the node distance d.

This procedure is used to generate simulation scenarios for arbitrary $n \times m$ intersections. Only even numbers for n and m are supported so that the number of incoming and outgoing lanes per road is equal. By omitting one of the outer nodes and adjusting the lane connections, T-junction versions with only three roads can be obtained. It suffices to consider only the removal of one specific node, e.g. the one at position $(0,d)$, because all other cases can be created by rotating a suitable network where only this node was removed.

The connections describing possible maneuvers are defined so that driving straight is allowed on all lanes and turning left or right is permitted only on the leftmost and rightmost lanes of a road, respectively. Turning directions on T-junctions are defined similarly for the two roads with matching directions, except that the connections to the missing road must be omitted. The lanes on the single, perpendicular road are split into left- and right-turning lanes in the middle; if the number of lanes is odd, the middle lane can be configured individually. In case a road has only one lane, all three crossing maneuvers are possible. For roads with more than one lane, dedicated left-turning lanes can be assigned by removing the possibility of driving straight on the leftmost lane. U-turn maneuvers are not supported.

To simplify the notation of specific intersection configurations, I introduce a straightforward naming scheme: Regular $n \times m$ intersections, i.e., those with four roads and no dedicated left-turning lanes, are denoted by `nxm`. For example, an unmodified 4×6 intersection scenario is uniquely identified by the name `4x6`. For the T-junction version of an intersection, the suffix `_T` is appended to its name. Similarly, the version with left-turning lanes is indicated by the suffix `_L`. Using this notation, a 6×6 T-junction is identified by the name `6x6_T`, a 4×6 intersection with left-turning lanes is called `4x6_L`, and a 6×8 T-junction with left-turning lanes has the unique name `6x8_T_L`.

There are some scenario generation parameters that have the same value for all scenarios used for the simulation studies. Most of them are required by SUMO for generating detailed traffic network descriptions with concrete shapes for all network elements. Starting with parameters affecting these physical shapes, all lanes of incoming and outgoing roads have the default width of 3.2 m. A distance value of $d = 175$ m is used to ensure that all vehicles can accelerate to the maximum allowed speed before starting the coordination procedure. The center lines of internal lanes are generated with 100 points so that the shape approximations used for the static data structure calculation are sufficiently precise (cf. Section 3.2.2). The width w of the internal lanes is set to 2.5 m. This value is sufficient to contain the exact maneuver shapes while avoiding unnecessary conflicts between internal lanes. For the computation of clearance positions, a step size of 0.05 m is used.

SUMO also defines speed limits for all lanes. For the incoming and outgoing roads, a maximum speed of 13.89 m/s ($\approx$50 km/h) is used, fitting the urban scenario. The speed limits on internal lanes for turning maneuvers are reduced automatically by SUMO to enforce realistically safe driving. The values are based on the turning radius and, therefore, differ between scenarios and even individual lanes. To give an example, the permitted turning speeds for a `4x4` intersection are 9.29 m/s for all left turns and 6.53 m/s for all right turns.

4.2 Traffic Generation

For a given intersection scenario, traffic is generated by scheduling a certain number of vehicles and assigning an individual route to each vehicle. The life cycle of a vehicle is defined as follows: SUMO spawns the vehicle into the simulation as soon as the scheduled start time is reached and there is enough room at the beginning of its start lane that it can be inserted without causing a collision. Every vehicle starts with an initial speed of 0 m/s. The first controller trigger is scheduled one trigger interval after the insertion time and the controller will thereafter be triggered continuously with this interval for the entire life cycle. The vehicle then approaches the intersection according to SUMO's default control systems, keeping a safe distance to the surrounding vehicles and driving as quickly as possible and permitted. As soon as the crossing controller has managed to place a reservation, the vehicle performs its crossing maneuver and continues driving on the outgoing target lane. When it reaches the end of this lane, the vehicle is removed from the simulation. Its *trip time* is then defined as the time that has passed since the actual insertion time. The simulation ends after all vehicles have left the scenario or when a time limit is exceeded. This constitutes one *simulation run*.

As the vehicle life cycle shows, the route definition of each vehicle comprises only the minimum insertion time as well as a start and target lane that uniquely identify the crossing maneuver. These routes are generated randomly and are based on the intersection structure and two traffic generation parameters: The *traffic demand D* and the *insertion time frame T*. The traffic demand parameter D controls the vehicle insertion rate and is specified as the number of vehicles entering the simulation per hour and per lane (veh/(h lane)). The insertion time frame T controls the duration over which the vehicle's insertion times are distributed in seconds. Given the number n_i of incoming lanes in the scenario, the total number of vehicles N is computed as

$$N = \left\lfloor \frac{D}{3600} \cdot T \cdot n_i \right\rfloor. \tag{4.1}$$

The minimal insertion times of all N vehicles are distributed uniformly at random in the interval $[0, D]$. Note that a simulation run will always simulate at least D seconds plus the minimum trip time of a vehicle, unless a time limit is reached first. Similarly, each vehicle is randomly assigned one of the n_i incoming lanes as its start lane. Once the start lane of a vehicle is fixed, the set of possible target lanes depends on the permitted crossing maneuvers defined by the intersection structure. If the lane only allows one maneuver, the target lane of this maneuver is the only choice. If multiple crossing maneuvers are permitted, the target lane is again selected randomly. However, the probabilities for all turning directions can be configured for each combination of possible maneuvers. For example, incoming lanes that allow driving straight and turning left could have a probability of 20 % for turning, while lanes that allow driving straight and turning right could have a 30 % chance of turning. This provides limited control over the overall distribution of turning maneuvers in the generated traffic.

Table 4.1 provides several example configurations of the probabilities assigned to each turning combination and the resulting absolute turning probabilities among all vehicles. The maneuver directions straight, left, and right are denoted by the letters s, l, and r. For a given combination c of possible directions, the probability to select the turning direction d is called $P(d \mid c)$. Observe that for the unique slr combination, two probabilities must be defined.

Configured probabilities			Absolute probabilities		
$P(\mathrm{l} \mid \mathrm{sl})$	$P(\mathrm{r} \mid \mathrm{sr})$	$P(\mathrm{l}/\mathrm{r} \mid \mathrm{slr})$	$P(\mathrm{s})$	$P(\mathrm{l})$	$P(\mathrm{r})$
0	0	0/0	1	0	0
1	1	0.5/0.5	0	0.5	0.5
1	0	1/0	$0.\overline{3}$	$0.\overline{6}$	0
0.1	0.1	0.05/0.05	0.9	0.05	0.05
0.2	0.3	0.2/0.3	$0.\overline{6}$	$0.1\overline{3}$	0.2

Table 4.1 – Distribution of turning maneuvers in a 2x4 scenario for different probability configurations. The 2x4 scenario has two lanes for each of the combinations sl, sr, and slr (cf. Figure 4.1b). This enables a configuration like the one shown in row 2, where all vehicles perform turning maneuvers and none drive straight. Note that it is not possible to make all vehicles turn in the same direction, e.g. left, because the traffic is distributed evenly over all lanes and the two sr lanes do not allow turning left (cf. row 3). The setting $P(\mathrm{l} \mid \mathrm{sl}) = 1$ effectively turns the intersection into the 2x4_L version.

To calculate the absolute probability of a vehicle performing a crossing maneuver in direction d, let C denote the set of direction combinations that include d, and n_c for $c \in C$ denote the number of lanes in the scenario that permit exactly the combination c. The absolute probability is then calculated as

$$P(d) = \sum_{c \in C} \frac{n_c}{n_i} \cdot P(d \mid c). \tag{4.2}$$

For example, the probability to turn right for the configuration shown in the last row of Table 4.1 is

$$P(\mathrm{r}) = \frac{2}{6} \cdot P(\mathrm{r} \mid \mathrm{sr}) + \frac{2}{6} \cdot P(\mathrm{r} \mid \mathrm{slr}) = \frac{2}{6} \cdot 0.3 + \frac{2}{6} \cdot 0.3 = 0.2, \tag{4.3}$$

with $C = \{\mathrm{sr}, \mathrm{slr}\}$, $n_{\mathrm{sr}} = n_{\mathrm{slr}} = 2$, and $n_i = 6$.

Just like the intersection scenario generation, the definition of simulated traffic requires a number of additional parameters and settings that are independent of the way individual routes are scheduled and that always have the same values. Most importantly, the vehicle type needs to be specified. For all simulations, every vehicle has the default vehicle type of the used SUMO version, which has a maximum acceleration of $2.6\,\mathrm{m/s^2}$, a maximum deceleration of $4.5\,\mathrm{m/s^2}$, and a rectangular shape that is 5 m long and 1.8 m wide. Because the speed limit on all lanes in every network is at most 13.89 m/s, this is used as the maximum speed of the vehicles.

Another important simulation setting is the *car-following model* used by SUMO to control the desired speed, safety gap, and acceleration of each vehicle in each time step. Here, SUMO's standard system that implements the car-following model presented by Krauß [23] is used. The exact implementation differs slightly from the original specification because the developers of SUMO applied slight modifications in order to preserve its safety properties in the simulation environment.

Finally, the definition of seeds for the random number generator that is used for traffic generation (in the following called *traffic seeds*) should be mentioned. It is designed to be independent of the seed used by the simulation framework at runtime to allow for multiple repetitions of the same simulation settings with the same traffic for different runtime seeds. The reason for this is that the traffic seed generally has a much larger impact on the outcome of the simulation because two different traffic seeds lead to completely different routes that could potentially interact in vastly different ways.

4.3 Simulation Parameters

Veins offers a multitude of configuration options to simulate the different aspects of communicating vehicles at various levels of detail and realism. Especially the modeling of the wireless communication can have a great influence on the simulation outcome since the coordination protocol relies on the vehicles having sufficient knowledge about the traffic situation at the intersection. Apart from that, certain idealizing assumptions can be broken by introducing random errors and deviations in order to simulate the vehicle behavior under less than perfect conditions.

This section presents the main simulation parameters used to control the realism aspects of the performance evaluation. Additionally, it provides an overview of the remaining parameters that are relevant for analyzing the simulation results.

Communication Model

As outlined in the introduction to the simulation framework (Section 2.3), the network simulator OMNeT++ provides mechanisms for sending messages via highly customizable channels, and Veins provides several models for simulating wireless communication using this functionality. These facilities are used to define a number of communication models that simulate different levels of realism.

The most basic communication model does not use any of the aforementioned features. Instead, it simulates *perfect communication* without errors, delay, or loss of information, i.e., messages arrive in the same simulation time step in which they are sent. This model is used for validation purposes and serves as a baseline for evaluating more realistic models.

The perfect communication model is a special case of the *unit disk* model. The unit disk communication model has an additional *communication range* parameter r

that specifies the maximum range at which two vehicles can communicate in meters. The distance between two vehicles is measured as the distance between the center points of the rectangles that represent their shapes. Whenever a broadcast message is sent, all vehicles with a distance $\leq r$ to the sender receive the message instantly and without errors, just as before. All other vehicles do not receive the message. The perfect communication model can be seen as a unit disk model with $r = \infty$. This model is useful for simulating a limited communication range that could be caused by attenuation effects on wireless signals, for example.

Veins provides the *simple path loss* communication model that utilizes the simulated wireless communication stack and implements a free-space path loss model with additional pass loss exponent α. A value of $\alpha = 2$ corresponds to the regular Friis path loss formula, larger values lead to stronger attenuation, and smaller values to less attenuation. Typical values for modeling outdoor environments range from $\alpha = 2$ to $\alpha = 4$. The effects of this model are a signal attenuation that increases with the square of the distance between the sending and receiving vehicles and a small delay simulating the time that a message requires to reach the receiving vehicle when traveling at the speed of light. The attenuation is used by Veins to calculate probabilities for losing messages; naturally, the probability for not receiving a strongly attenuated signal is larger.

Another wireless model from Veins is the *obstacle shadowing* model. It was introduced by Sommer et al. [24] to model attenuation effects of obstacles such as buildings in the line of sight between the sending and the receiving vehicle. The model was developed to work with the Veins IVC stack and was validated against measurements from real-world experiments. In the simulation, buildings are simple polygons and the attenuation strength is calculated based on the number of times a signal passes through the walls of buildings and the distance it travels inside of the buildings. These two factors are parameterized so that the simulation can be configured to model multiple building types with different attenuation characteristics.

However, this is meant to be used for calibrating the model for very specific buildings and it is difficult to define suitable settings for a wide range of general obstacle conditions. Instead, the obstacle shadowing model is adopted for the simulations as follows: The attenuation parameters are configured such that buildings block all messages that travel through them. The intersection scenario generation is extended to add a square-shaped building between each pair of neighboring roads, as depicted in Figure 4.2. Instead of using a configurable attenuation strength, the distance d_B between the buildings and the roads is used as a simulation parameter to control the influence that buildings have on the communication. With a side length of 100 m, the buildings are large enough that even for small values of d_B, vehicles at the outer ends of neighboring lanes cannot communicate.

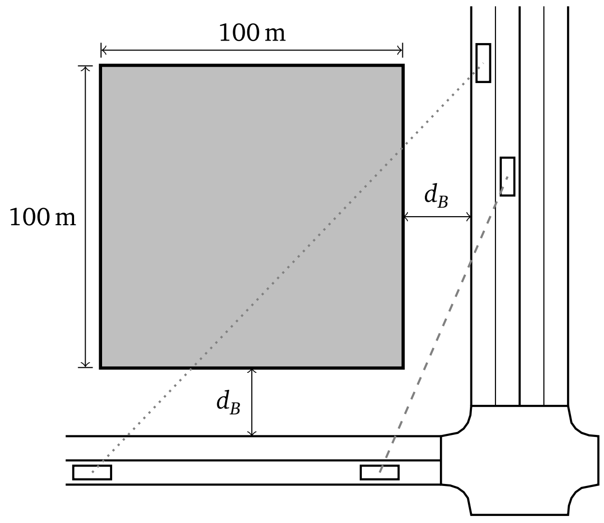


Figure 4.2 – Placement of buildings in the simulation scenario. Buildings are placed the same way on all four sides of the intersection. The dotted line shows the line of sight between two vehicles whose communication is blocked by the building. The vehicles connected by the dashed line can exchange messages without being affected by the building.

Both the simple path loss and the obstacle shadowing model use the simulated IVC stack implementation of Veins. This means that both introduce a propagation delay proportional to the transmission distance, and a small processing delay that simulates the time required by the lower communication layers to perform the transmission on the sending and receiving vehicles. Because of the small size of the coordination messages and the relatively small communication distances in the simulation scenarios, these effects usually amount to delays of less than 1 ms between the vehicle applications of the sender and receiver sending and processing the message, respectively. Nevertheless, they contribute to the realistic simulation of wireless communication. Veins also allows to use both of these models at the same time so that messages that are not blocked by a building are still affected by attenuation from free space path loss. This is the most realistic configuration of the communication model parameters.

Processing Delay

The implemented communication protocol, consisting of only four messages which are always broadcast and neither validated nor answered, is extremely simplistic. Because a reliable communication is generally essential for the safety of a distributed coordination system, real implementations of such applications feature sophisticated communication protocols to ensure maximum reliability. Such protocols typically have to meet various requirements and accomplish this by exchanging more information, sending messages more frequently, and forming flexible networks as communication infrastructure [25]. Common techniques for ensuring high reliability are acknowledgment messages and repeat requests, while other requirements like security, for example, are met with more protocol adjustments like mechanisms for authentication and encryption. All of these additional factors contribute to communication and processing overhead, ultimately leading to a larger communicational delay.

To take this into account in the simulations, a *processing delay* parameter t is introduced. It is implemented by making the vehicle application wait t seconds before processing a received message. The application is not aware of any messages that are currently being processed because this would require significant modifications of the controller implementation that would likely cause changes of the vehicle behavior and obscure the effects of the processing delay. Note that this delay is only applied *after* a message was successfully received. This means that all delay introduced by the communication model, i.e., the simple path loss and obstacle shadowing models, still occurs if one of these models is used, even if the processing delay is set to $t = 0$.

Sensor Error

The intersection coordination protocol uses positional information of other vehicles to decide whether a planned maneuver is safe. More specifically, the *potential collision check pc* uses the lane positions of vehicles on conflicting lanes to detect potential collisions (see Section 3.3). Therefore, the way this information is acquired is relevant to the safety of the protocol.

The implemented traffic model interface provides direct access to the positions of other vehicles, just like the assumption of perfect sensor information in the abstract model suggests. In reality, however, on-board sensors of vehicles cannot be expected to display such accuracy. Because the lane positions of other vehicles are perceived directly through the ego vehicle's sensors instead of being communicated like the claims and reservations, this aspect is not affected by the communication model and processing delay parameters. To simulate the inaccuracy of the sensor information, a *sensor error* parameter e is introduced. Whenever the vehicle application requests positional information of another vehicle, a random error is added to the real value by the traffic model interface. The error value is normally distributed with mean 0 and standard deviation e. A standard deviation of $e = 0$ denotes the perfect information setting. Thereby, over- and underestimations of the actual vehicle positions are equally possible and small errors are more probable than large errors, which are reasonable properties to assume.

Artificial Message Loss

In addition to the realistic wireless communication models that can already cause the loss of messages, a separate, artificial *message loss* parameter p is introduced. Here, p simply specifies the probability for ignoring a message that was received successfully. A setting of $p = 0$ has no effect while $p = 1$ leads to all messages being blocked, effectively eliminating all communication possibilities of the vehicles. The purpose of this parameter is mainly the isolated investigation of the effect that missing information has on the controller's safety performance. In this regard, its advantage over the realistic communication models is that it is completely independent of the relative vehicle positions and the structure of the simulation scenario.

Other Parameters

There are several other relevant simulation parameters that are not as interesting for the performance evaluation but should be addressed nevertheless. First of all, the simulation step size of SUMO is set to 0.1 s. Thereby, all vehicle positions are updated ten times per simulated second. This value is small enough that the vehicles' trajectories appear smooth and their microscopic behavior can be analyzed, but not so small that it significantly decreases the simulation speed. Note that OMNeT++ still simulates all events at their own, arbitrary time scales; vehicles just do not change their positions between two SUMO time steps.

Unless stated otherwise, the controller trigger interval is also set to 0.1 s. This is a reasonable rate of sending coordination messages because safety-critical information should be exchanged frequently and modern hardware can offer the necessary performance. Real implementations of communication protocols often demand even higher frequencies around 50 Hz [25] and experiments show that state information updates at the order of 10 Hz can suffice for successful collision avoidance [14], [15]. However, a trigger frequency that is higher than the frequency of position updates is not really sensible because it would lead to multiple triggers for the same vehicle positions.

The crossing controller uses three constants d_c, t_c, and t_{cr} that affect the vehicles' behavior and require appropriate values. d_c specifies the distance to the intersection at which a vehicle starts the coordination procedure and is used by the *crossing ahead check* ca(ego). Given that the vehicle characteristics and scenario structure are known, a value of $d_c = 30$ m seems reasonable: The maximum allowed speed on the intersection's incoming lanes is 13.89 m/s. Assuming that the vehicle continues to travel at this speed when it comes within coordination range, it still has $30/13.89 \approx 2.16$ seconds before its safety envelope would enter the intersection and it would be forced to brake if it did not place a reservation by that time. This time should be more than sufficient to negotiate a maneuver because the controller is triggered at least 20 times and usually requires at most 4 triggers to place a reservation if no other vehicles cause potential collisions. If, however, no reservation can be placed, the vehicle will have to slow down. In this case, there is still enough room to do so using the regular, smooth deceleration rather than applying the full brake power. Furthermore, at maximum speed, the braking distance is approximately $(13.89)^2/(4.5 \cdot 2) \approx 21.4$ m (recall that the maximum deceleration is $4.5\,\text{m/s}^2$). This means that vehicles will be within $30 + 21.4 = 51.4$ m of the intersection when sending their first coordination message, which keeps the communication range requirement relatively low.

Finding suitable values for the time thresholds t_c and t_{cr} is more straightforward. t_c specifies the maximum time a claim can be active until it must be turned into a reservation or withdrawn if the former is not possible. It was originally introduced as a precaution to avoid deadlocks [1] and is used in the implementation for the same purpose by deleting outdated traffic information after some time in case an `rc` or `wd_cc` message is not received. To give the vehicle that sent this message sufficient time to finish its maneuver, a value of $t_c = 5\,\text{s}$ is used. This value is large enough that many controller triggers can occur and vehicles can move significant distances so that it is safe to delete the claim information.

Similarly, t_{cr} specifies the maximum time a reservation may be active before it has to be withdrawn. It has the same purpose and is implemented the same way as t_c. However, this value can be determined more accurately by calculating the worst-case duration of a crossing maneuver because all lengths of internal lanes, speed limits, and the maximum acceleration value are accessible. Simulating a vehicle that performs the longest possible crossing maneuver, starting at the maximum distance from the intersection according to d_c and accelerating from $0\,\text{m/s}$, yields a reasonably accurate estimation of the maximum crossing maneuver duration. Among the investigated simulation scenarios, no crossing maneuver takes significantly more time than $6\,\text{s}$. Therefore, a threshold of $t_{cr} = 7.5\,\text{s}$ seems to be a reasonable upper bound. Note that increasing this threshold should not be detrimental to the safety of the protocol because it only makes vehicles that have not received a relevant `wd_rc` message wait a few seconds longer.

4.4 Performance Metrics

To assess the performance of the crossing controller by analyzing the simulation output, suitable metrics are required. These metrics are focused on the safety aspects because safety is the main promise of the controller and, therefore, the measure by which it is evaluated. Other metrics for indicating efficiency are also considered to gain a better understanding of the simulated system.

Collision Count

The *collision count* is the most straightforward safety metric. It is determined by counting the physical collisions that occur during a simulation run. Collisions are found by overlapping the vehicles' rectangular shapes in each SUMO time step. If a collision between the same vehicles is detected in multiple consecutive time steps, it is only counted once. Recall that no further coordination actions are taken by vehicles that have already started their crossing maneuver; colliding vehicles will simply pass through each other and finish their maneuver as if nothing happened. The total collision count can be averaged over the number of vehicles to obtain a relative collision rate. This is useful for comparing simulation runs with different traffic demand parameters.

Note that if the crossing controller works as intended, there should not be any collisions at all. Therefore, this metric is most useful for finding simulation parameter thresholds at which the first collisions occur and estimating the impact of parameter values beyond these thresholds. The occurrence of the first collision is the point at which the controller is definitely not working properly and cannot provide the same level of safety as in the abstract model.

Relative Velocity

When a collision is detected, the *relative velocity* of the colliding vehicles can be determined. To this end, the velocity of each vehicle is represented by a two-dimensional vector that points in the vehicle's driving direction and whose length is the vehicle's current speed. The relative crash velocity is then computed as the length of the difference between the two vehicles' velocity vectors, as illustrated in Figure 4.3.

In order to increase the impact of high-speed collisions on the metric's value, each recorded relative velocity can be squared. This is reasonable to do because the kinetic energy of a moving object is proportional to the square of its velocity, making it feasible to use this metric as a measure of crash severity. The squared and accumulated crash velocity was also used by Dresner and Stone [12] to evaluate the effect of their safety measures in their simulations. Similar to the collision count, the accumulated relative velocity can also be divided by the number of vehicles to obtain a quantity that is comparable across simulation runs. and can be interpreted as a kind of risk for each vehicle. Taking the average (squared) velocity over all collisions instead provides another perspective that is independent of the number of vehicles and estimates the overall severity of collisions in the scenario more generally.

The relative velocity metric is only useful if there are collisions to begin with. However, it is more informative than the plain number of collisions and can be used to gain a deeper understanding of the collision characteristics, enabling a better

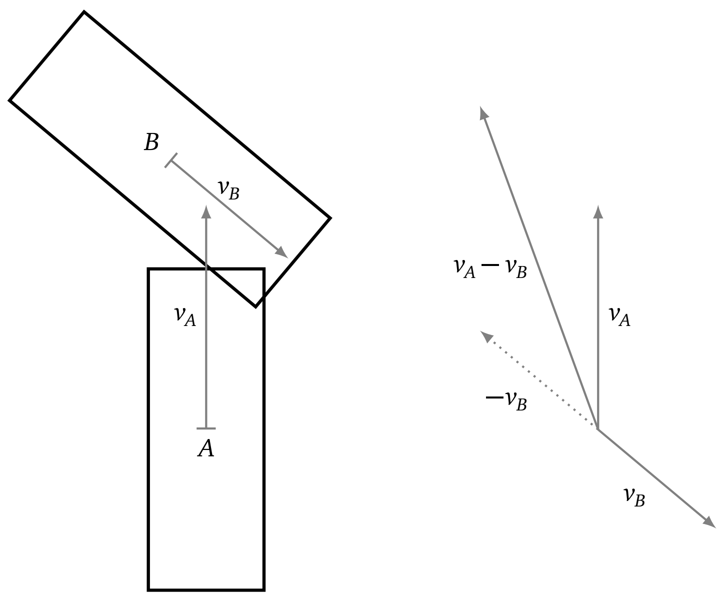


Figure 4.3 – Determining the relative velocity of a collision. Vehicles A and B collide while driving at speeds $||v_A||_2$ and $||v_B||_2$, respectively. The relative velocity is then determined as $||v_A - v_B||_2$. This calculation is performed in the first time step the collision is detected.

evaluation of the coordination protocol's weaknesses and their causes. Additionally, it might be useful for comparing simulation results where the total collision counts are very similar.

Reservation Conflicts

For evaluating the controller's performance without the need for actual collisions, the *reservation conflict* metric is used. Basically, a reservation conflict occurs when two vehicles have active reservations on conflicting internal lanes at the same time. It can be detected in the time step in which the second vehicle places its reservation and it exists until the first vehicle withdraws its reservation. During this time, the vehicles may or may not collide; outside of it, however, collisions are impossible by definition of the foe matrix that specifies the lane conflicts. Therefore, fewer reservation conflicts generally imply fewer opportunities for collisions.

However, the metric can be differentiated further to provide more meaningful information: The coordination protocol allows reservation conflicts to occur when one of the vehicles has passed its respective clearance position. In this case, a collision is impossible and it is safe for the second vehicle to place its reservation and start the crossing maneuver. This is classified as a *weak reservation conflict*. Weak conflicts do not threaten the safety of the involved vehicles; on the contrary, they can be indicative of a precise, well-timed coordination and reduce the waiting times of the vehicles.

If, on the other hand, both vehicles are still ahead of their respective clearance positions, a collision cannot be ruled out and a *strong reservation conflict* is registered. The occurrence of a strong conflict is evidence that the coordination protocol does not work as intended because it is a direct violation of the safety property $Safe^*$ as defined in Equation 3.5. This is convenient and extremely useful for the evaluation because it directly corresponds to the safety property that was proven to hold for the abstract controller.

Using these concepts for safety estimation yields various interesting metrics, especially when they are used in combination. For example, the relative frequency of weak/strong conflicts among all reservation conflicts can indicate how precisely crossing maneuvers are coordinated, while the total number of strong conflicts shows how often the protocol is violated. Relating the number of strong conflicts to the number of collisions can provide an estimation of how dangerous such conflicts are; if the risk for a collision in case of a strong conflict is very high, this could be a serious weakness of the coordination protocol. All of this is possible because of the hierarchical ordering of the safety metrics: If no reservation conflicts occur, all maneuvers are perfectly safe. Occurring conflicts are categorized into weak and strong conflicts, of which only strong conflicts allow collisions. Detected collisions can, in turn, be evaluated and compared by analyzing the (squared) relative velocity.

Other Metrics

In addition to the safety metrics, a number of other simulation outputs that are mainly related to efficiency are considered. They provide useful information for finding the causes of specific effects introduced by the simulation parameters and understanding the overall vehicle behavior.

To start with, the *trip times* of all vehicles in a simulation run are recorded. Because the exact lengths of the routes of all vehicles are different, the individual times and their average value are not really meaningful; the average trip time can only serve as a rough estimation of the intersection efficiency. However, the minimum and maximum times as well as the standard deviation can be useful for detecting irregular behavior and profiling the trip characteristics.

For observing the communication behavior, all sent and received coordination messages are logged. Message traffic is an important efficiency metric because in VANET research, keeping the wireless traffic low to avoid channel congestion and interference is typically one of the major challenges. Again, statistic values like the minimum and maximum number of sent or received messages are likely to be more useful than individual values. The coordination messages are more directly related to the vehicle controller behavior than trip times and can provide more detailed information on the controller's reaction to imperfect conditions.

Chapter 5

Evaluation

Both the controller implementation process and the subsequent simulation studies reveal various advantages and disadvantages of the abstract approach. While the findings of the implementation process show that the abstract controller definition makes it very adaptable but also introduces its own challenges, the simulations focus more on the influence of realism aspects and indicate that lifting the idealizing assumptions has a negative effect on the controller's safety.

In this chapter, I present and discuss these results in detail, focusing on the simulation studies. Section 5.1 provides a summary of the practical problems encountered during the implementation process and draws conclusions about their implications. Section 5.2 presents the simulation studies and analyzes their results to assess the impact of the parameters on the controller's performance. This is the main part of the chapter.

5.1 Implementation and Abstraction Level

The implementation of the crossing controller and its traffic interface revealed a number of design challenges and practical problems that are not only relevant for the simulations but also in the context of a real-world implementation.

The abstract nature of the controller definition leaves a large amount of design freedom as it does not specify execution rules, a communication protocol, or actual vehicle control inputs. The coordination protocol and controller behavior, however, are rather clear and intuitive. This makes it easy to design a control system and adapt the controller to work with existing components, facilitating a flexible, modular architecture.

But the abstraction level in combination with the precise, mathematical formalization of the system also introduces some intricacies that can cause problems when they are not detected and dealt with. For the set of concrete implementation design decisions made in this thesis, these problems manifest themselves mostly in blocked intersections, as presented in Section 3.4. It is likely that a different set of decisions in another application scenario would lead to other problems having different effects. However, the practical problems discussed here are representative of a possibly larger group of similar complications that reveal some inherent weaknesses of the abstract controller definition.

Starting with the livelock problem that was caused by non-atomic access to the crossing segments, the first of these weaknesses becomes obvious. It is clear that this behavior is closely related to the implemented execution model and would have occurred less frequently or not at all if a different model had been implemented. In this case, the design freedom is actually problematic because depending on the decision made in the implementation process, the controller may not work as expected, even if the employed execution model handles the automaton semantics correctly. Therefore, the lack of an execution model in the abstract controller definition, which is the reason for the design freedom, represents a major disadvantage of the abstract model.

Furthermore, this is part of a more general problem: Because there is no concrete execution model defined, the safety property of the controller formally holds for any model that adheres to the automaton semantics. Thus, it is impossible to detect problems such as the aforementioned livelock within the abstract model, where the vehicle controller is originally defined. This shows that additional properties besides safety, such as liveness, are not only favorable but necessary in order to take advantage of the benefits of the abstract specification approach.

The *on crossing check* $oc(\text{ego})$ also caused an implementation problem that has a similar origin. In addition to the execution model, the precise formalization of the UMLSL formulas and their interpretation are involved here. Recall that the problem with the *on crossing check* was caused by the formula $oc(\text{ego})$ being a condition for placing a reservation but also requiring the ego vehicle to have entered the intersection to be satisfied. Because in the simulation environment, vehicles have to place their reservations *before* entering the intersection, the requirements formed a cyclic condition that could never be satisfied.

In the abstract model, a dedicated *distance controller* is responsible for setting vehicle control inputs and ensuring that vehicles do not enter an intersection without a reservation. However, it is not defined explicitly and just assumed to work properly; its behavior in this specific situation is not obvious but can be assumed to allow entering the intersection the moment the reservation is about to be placed. This is an intricacy of the mathematical formalization that does not translate well into less abstract situations with a concrete execution model. The problem with the *on crossing check* itself is not very serious and was fixed in the implementation easily, but it represents another group of problems to be aware of when implementing a controller defined in the abstract model.

The two last problems discussed in Section 3.4 occurred when coordination messages were received with a delay or were lost completely. Obviously, the lack of a communication model and mechanisms for dealing with unreliable information is the main reason why such events cause problems. Here, it is the developers' job to implement a suitable communication protocol that ensures timely and reliable traffic updates. This freedom of choosing a communication protocol and designing an appropriate representation of the relevant traffic and coordination information is a very useful property of the abstract model and can, evidently, be used to solve these two problems. However, defining such mechanisms is essential because the controller itself is very susceptible to delayed or missing information; in both cases, the observed result was a completely blocked intersection.

To summarize this discussion, one can say that the great freedom in designing the components of a concrete implementation of the controller has both positive and negative effects. In general, the high level of abstraction facilitates a modular design that is easy to understand and can incorporate already existing solutions for specific tasks, such as a communication protocol. However, the abstract model has its intricacies that can cause problems if they are not considered. Especially the lack of an execution model for the controller automaton can be problematic to the point where the automaton itself has to be adjusted in order to make it work properly. In most of these cases, though, it is possible to find a solution that is still reasonably close to the original abstract definition.

5.2 Simulation Results

Having discussed and resolved the implementation issues, the controller's performance can be evaluated through simulations. To this end, I conducted extensive simulation studies to investigate the effects of the previously introduced parameters. The results of these simulations are presented and evaluated in this section, starting with a baseline configuration that represents the least realistic scenario and that was also used for validation purposes. Afterwards, each of the parameters is studied individually in order to assess their impact independently of each other and of possible interactions. Finally, combinations of selected parameter settings are investigated to find such interactions and compare the severity of individual effects.

5.2.1 Individual Parameters

First of all, an appropriate selection of simulation scenarios must be defined. Because urban intersections are highly diverse, these scenarios should cover a wide range of intersection layouts. The simulation scenarios in most related studies focus on symmetrical four-way intersections without dedicated turning lanes [11]–[13], [16], [17]; the majority of these consider intersection sizes between 2×2 and 6×6. Using the scenario generation method described in Section 4.1, it is possible to create arbitrary $n \times m$ intersections with optional left-turning lanes as well as their T-junction variations. Additionally, the probabilities for turning maneuvers can be configured to some extent. In the related studies that allow turning maneuvers and provide turning probabilities, the absolute probabilities are specified as 90 %/5 %/5 %[12], 66.6 %/16.6 %/16.6 %[13], and 70 %/20 %/10 %[17] for driving straight/right/left, respectively. Assuming that these probabilities represent somewhat realistic urban traffic behavior, the simulation scenarios here are configured to use similar turning probabilities. A list of all scenarios with relative and absolute probabilities is provided in Table 5.1.

	Configured probabilities			Absolute probabilities		
Scenario	$P(\mathrm{l} \mid \mathrm{sl})$	$P(\mathrm{r} \mid \mathrm{sr})$	$P(\mathrm{l/r} \mid \mathrm{slr})$	$P(\mathrm{s})$	$P(\mathrm{l})$	$P(\mathrm{r})$
2x2	-	-	0.15/0.15	0.7	0.15	0.15
4x4	0.3	0.3	-	0.7	0.15	0.15
6x6	0.45	0.45	-	0.7	0.15	0.15
8x8	0.6	0.6	-	0.7	0.15	0.15
2x2_T	0.25	0.25	-	0.5	0.25	0.25
4x4_T	0.5	0.5	-	0.5	0.25	0.25
6x6_T_1	0.5	0.5	-	$0.\overline{5}$	$0.1\overline{6}$	$0.2\overline{7}$
6x6_T_2	0.5	0.5	-	$0.\overline{5}$	$0.2\overline{7}$	$0.1\overline{6}$
8x8_T	0.75	0.75	-	$0.541\overline{6}$	$0.2291\overline{6}$	$0.2291\overline{6}$
4x4_L	-	0.3	-	0.5	0.35	0.15
6x6_L	-	0.45	-	$0.51\overline{6}$	$0.\overline{3}$	0.15
8x8_L	-	0.6	-	0.6	0.25	0.15
2x4	0.3	0.3	0.15/0.15	0.7	0.15	0.15
2x6	0.45	0.45	0.15/0.15	0.7	0.15	0.15
2x8	0.6	0.6	0.15/0.15	0.7	0.15	0.15
4x6	0.375	0.375	-	0.7	0.15	0.15
4x8	0.45	0.45	-	0.7	0.15	0.15
6x8	0.525	0.525	-	0.7	0.15	0.15

Table 5.1 – All considered simulation scenarios with their turning probability settings (see Table 4.1 for an explanation of the notation). There are two versions of the `6x6_T` scenario, one with the middle lane of the single, perpendicular approach turning right, and a second one with this lane turning left. The `2x2_T` scenario is the only one with an lr lane, which is why the column for $P(\mathrm{l} \mid \mathrm{lr})$ is omitted. Its value for the `2x2_T` scenario is 0.5. Note that a `2x2_L` scenario does not make sense because all vehicles would have to turn left. The turning probabilities in all scenarios are defined to be as comparable as possible.

All of these scenarios were used for validation and for the baseline simulation. In this simulation, which represents the least realistic configuration, a perfect communication model with no additional parameters introducing errors or delay was used. The only simulation parameter was the traffic demand D, for which values from 200 to 800 veh/(h lane) in steps of 50 were simulated for each intersection scenario. These demand values are comparable to related work, where the maximum traffic demand ranges between 500 veh/(h lane)[16] and 1200 veh/(h lane)[12]. Note that most of these simulation studies were focused on efficiency in terms of intersection throughput and vehicle waiting time, wherefore their management systems could handle relatively large traffic volumes.

Each simulation run was repeated 200 times; two repetitions with different runtime seeds for 100 different random traffic seeds. The insertion time frame T was set to 120 s, i.e., the minimum vehicle insertion times were distributed evenly over the first two minutes of each simulation run. These were the default settings for all other simulations as well.

The main result of this simulation is quite simple: In the least realistic configuration, the crossing controller works exactly as intended and provides perfect safety. There were no recorded strong reservation conflicts or even collisions in any simulation run and all vehicles were able to perform their crossing maneuver, i.e., there were no blocked intersections. Additionally, the number of sent coordination messages was exactly three for all vehicles: One `cc` message to place the claim, one `rc` message to turn the claim into a reservation, and one `wd_rc` message to withdraw the reservation after finishing the crossing maneuver. There was also a relatively large number of weak reservation conflicts, indicating that the coordination and communication are precise enough to allow many maneuvers to start as soon as the respective internal lane is clear, even if another vehicle still has a conflicting reservation.

Figure 5.1 displays the average number of reservation conflicts per vehicle for all scenarios. Among the different intersection layouts, there is a general trend towards a higher number of conflicts for larger numbers of incoming lanes. This is not surprising because a larger number of conflicts between internal lanes does not only increase the total number of reservation conflicts (recall that more lanes lead to a larger total number of vehicles) but also the potential of any individual vehicle to be involved in multiple conflicts. Comparing the different scenario variations, it stands out that in all cases, the `nxn_L` version has a larger number of conflicts than the basic `nxn` scenario, while the `nxn_T` variants have the lowest conflict numbers among all scenarios. The reason for this is that left-turning maneuvers belong to the longest internal lanes and have the highest number of foes; both of these properties increase the potential for weak reservation conflicts involving left turns. On the contrary, the T-junction variants have a much lower number of conflicts among their internal lanes. In scenarios with more than one incoming lane per direction (i.e., all `nxn_T` scenarios except `2x2_T`), many of the straight maneuvers and one right turn

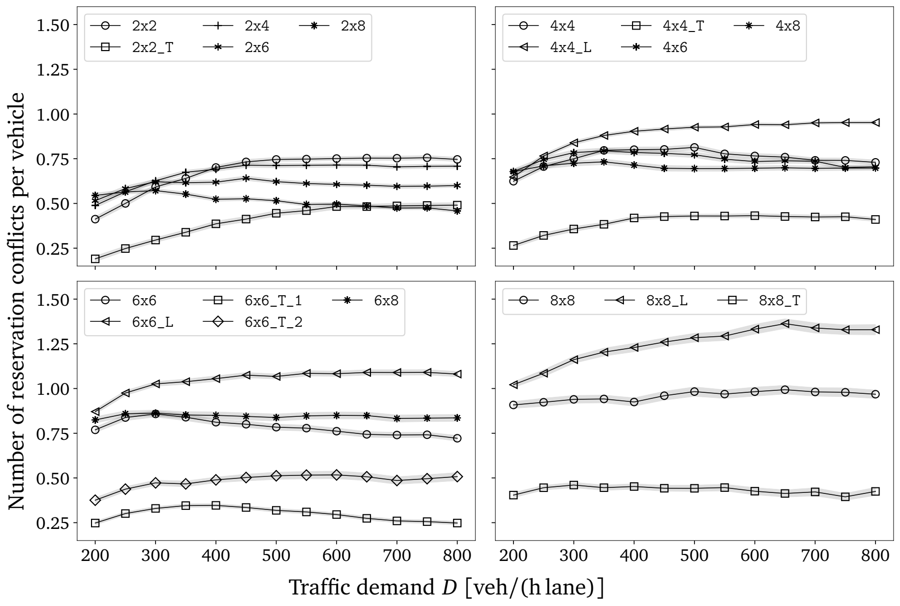


Figure 5.1 – Number of reservation conflicts per vehicle in the baseline simulation. Each data point represents the number of conflicts averaged over all vehicles and the 200 repetitions. The shaded areas around the graphs mark the 95 % confidence intervals. Note that all recorded conflicts are weak conflicts.

have no foes at all, which leads to a large fraction of vehicles that are never involved in any reservation conflict. The asymmetrical `nxm` scenarios are generally similar to the basic `nxn` versions but seem to cause fewer conflicts if the difference between `n` and `m` is large. This can be explained by the vehicle behavior: Vehicles on the minor approaches must wait relatively long for an opportunity to cross the intersection because the major approaches support many simultaneous maneuvers blocking the way. For example, straight driving vehicles on one of the single-lane approaches in the `2x8` scenario have to wait at least until all eight internal lanes belonging to straight maneuvers of the major approaches are clear, whereas the vehicles on these approaches are far less restricted. Naturally, waiting vehicles are not involved in any reservation conflicts, leading to a generally lower number of conflicts per vehicle. This effect is less significant for the `4xX` scenarios and vanishes in `6x8` due to the smaller difference between the approaches.

In all scenarios, the number of reservation conflicts initially increases with the traffic demand level and stagnates at the higher demand levels. This is because for low traffic demand, the intersection is sparsely populated with vehicles and the crossing maneuvers do not occur frequently enough to cause many weak reservation conflicts. Higher traffic demands have the opposite effect and lead to a saturation that prevents any further conflicts, sometimes even reducing the number. The demand level at which this happens is different for each scenario.

The baseline simulation shows that (1) the implemented controller functions as intended and (2) that the safety property of the abstract crossing controller definition translates into a more realistic scenario if the controller is implemented appropriately. While the first result (1) is mainly relevant for the purpose of validation and verification, result (2) is rather significant because it means that the abstract approach to define vehicle controllers with provable properties is viable, at least to some extent. However, the baseline scenario has the most abstract parameter configuration possible and it is only halfway between the abstract model and a highly realistic simulation setting. In order to progress towards such more realistic configurations, the individual simulation parameters are investigated next. For these simulations, only a representative subset of the intersection scenarios is considered, namely the `4x4`, `4x4_T`, `4x4_L`, and the `2x4` scenario.

To start with, the perfect communication model is just a special case of the unit disk model with unlimited communication range $r = \infty$. By limiting this range, vehicles lose the ability to communicate and coordinate with other vehicles depending on the distance between them. It is a highly unrealistic assumption that vehicles can communicate at arbitrary ranges, especially in the urban scenario where various obstacles can block the line of sight and the high density of transmitting agents can cause interference, in addition to attenuation effects in wireless transmissions.

As a first approximation of this, I ran simulations for the communication range parameter r, ranging from 0 m to 200 m. To investigate different levels of traffic volume, I repeated the simulations for demand values D of 200, 500, and 800 veh/(h lane). These values will henceforth be referred to as *low*, *medium*, and *high* traffic demand, respectively.

Figure 5.2 shows the average number of strong reservation conflicts per vehicle at medium traffic demand. The plots display a number of distinct steps that can be explained with several communication range thresholds: For range values r from 0 m to the first threshold at about 20 m, the vehicles cannot communicate sufficiently to avoid conflicts; they are practically unaware of each other's maneuvers. After this first threshold, the number of conflicts decreases rapidly because the vehicles are now capable of coordinating with other vehicles on neighboring approaches. Until the third threshold at about 55 m, an increasing number of vehicles on the neighboring roads gets within range, further decreasing the number of conflicts. Observe that the rate at which this number decreases varies in distinct phases. This is due to the individual lanes of the neighboring roads coming within range at slightly different values of r. The second threshold of about 35 m is roughly halfway between the other thresholds and marks the range value where all lanes on the neighboring approaches are at least partially in range. At 55 m, another sharp decline of reservation conflicts

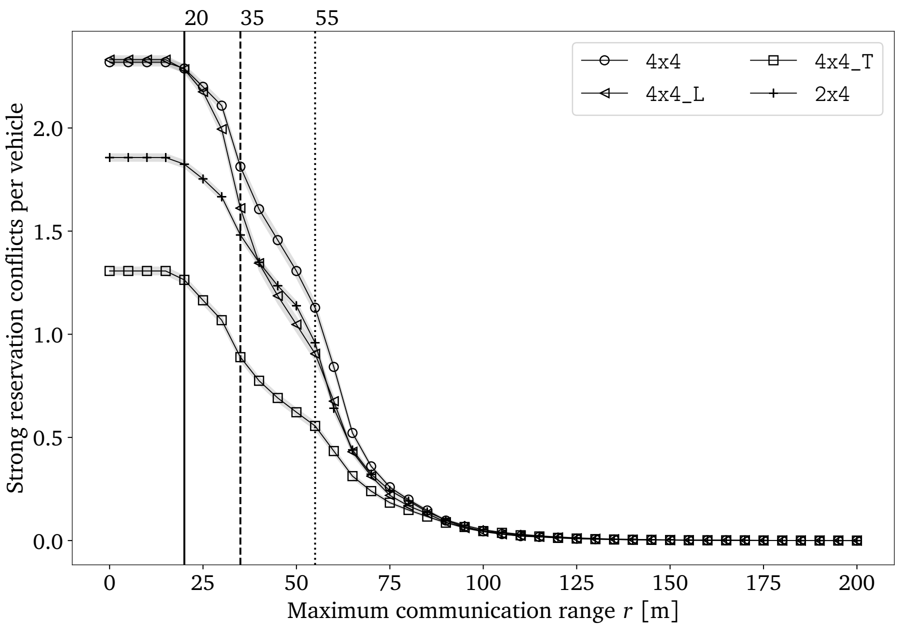


Figure 5.2 – Average number of strong reservation conflicts per vehicle for medium traffic demand ($D = 500\,\mathrm{veh/(h\,lane)}$). The solid, dashed, and dotted vertical lines mark the communication range thresholds at 20 m, 35 m, and 55 m, respectively.

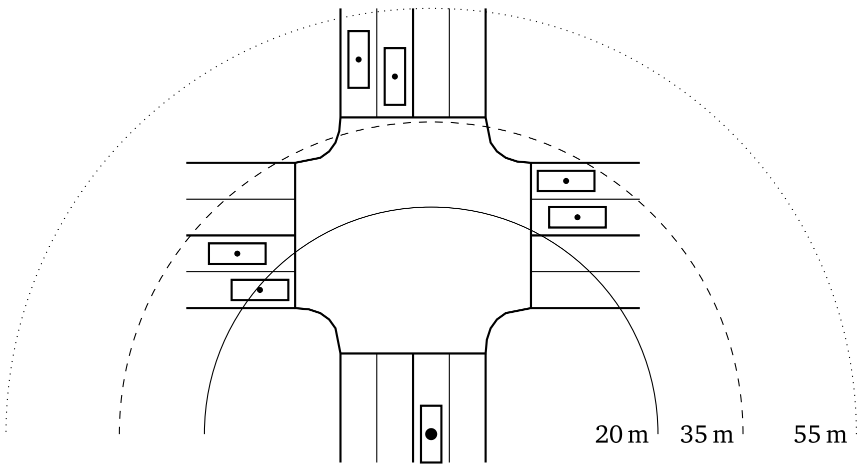
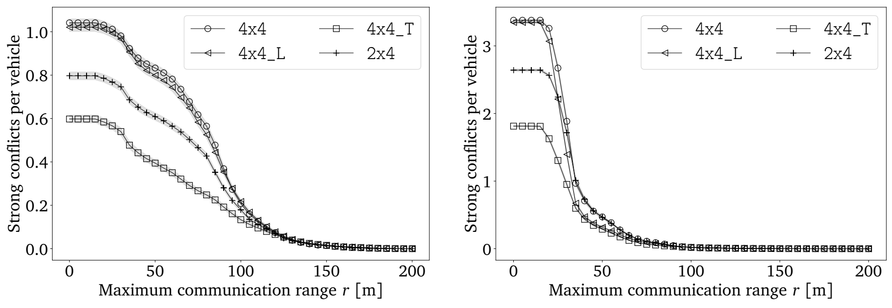


Figure 5.3 – Visualization of the rough communication range thresholds in the 4x4 scenario (not to scale). The solid, dashed, and dotted arcs correspond to the vertical lines in Figure 5.2.

is visible. This is the range at which it is possible to communicate with vehicles on the opposite side of the intersection. The number of conflicts then decreases until about 120 m, where it reaches zero. Because the required communication range strongly depends on the positions of the sending and receiving vehicles, these threshold values are only representatives of intervals in which smooth transitions between the aforementioned phases occur. Figure 5.3 depicts the 4x4 intersection with the approximate communication range thresholds.

For the other traffic demand levels, the overall results are very similar. However, the communication range at which no more conflicts occur is larger for low traffic demand and smaller for high demand, as shown in Figure 5.4. The reason for this is that for higher traffic demand, the intersection tends to fill up even if only a few vehicles manage to coordinate their maneuvers, leading to many waiting vehicles. These vehicles then require a significantly lower communication range in order to coordinate effectively so that even more vehicles wait at the intersection and the number of strong reservation conflicts becomes comparatively small.



(a) $D = 200\,\mathrm{veh/(h\,lane)}$ **(b)** $D = 800\,\mathrm{veh/(h\,lane)}$

Figure 5.4 – Strong reservation conflicts per vehicle – Comparison of low and high traffic demand.

Another interesting aspect is that the required communication range for safe coordination is well below its theoretical upper bound: The largest possible communication range requirement occurs when two vehicles A, B approach the intersection from opposite directions and travel at the maximum speed of 13.89 m/s. As outlined previously, the distance between the front bumper of a vehicle and the intersection in this situation is about 51.4 m; adding the offset of 2.5 m to the center point of the vehicle yields a distance of 53.9 m between A's center and the holding line of the intersection. The diameter of the intersection is roughly 20 m. In the worst case, the second vehicle B places its reservation very shortly before A finishes its maneuver. Note that this is only the worst case for a strong reservation conflict and a collision is very unlikely in this scenario. Assuming that A requires about 6 s for its crossing maneuver, B can travel at most $6 \cdot 13.89 = 83.34\,\mathrm{m}$ after the first reservation is placed. If B places its reservation at the first opportunity, i.e., 53.9 m from its holding line, it was at most $53.9 + 83.34 = 137.24\,\mathrm{m}$ away from the intersection when the first reservation was placed. Combining these distances results in a total distance of $53.9 + 20 + 137.24 = 211.14\,\mathrm{m}$. Therefore, A requires a communication range that covers at most this distance for B to receive the reservation message. The simulation results, however, suggest that even a range of about 125 m is already sufficient for safe coordination.

These first results show that the coordination protocol requires a certain minimum communication range to function reliably. The more vehicles with conflicting maneuvers are within range, the fewer reservation conflicts and collisions occur. The necessary range depends indirectly on the intersection dimensions, the traffic demand, and the permitted speed; starting the coordination procedure at a smaller distance to the intersection naturally lowers the required range. This could also be influenced by lowering the value of the d_c constant but this would eventually force the vehicles to slow down before entering the intersection, even if there is no need for coordination.

The *simple path loss* model of Veins calculates the free-space path loss according to the transmission distance and the path loss exponent α. In the Veins IVC stack model, packets are lost if the received transmission power falls below a certain threshold. Therefore, the simple path loss model effectively limits the communication range of vehicles in a similar way to the unit disk model because the received power decreases with increasing distance. However, it also introduces a propagation and processing delay and provides a much more realistic model of real wireless communication technology than the simple unit disk model which does not use the IVC stack.

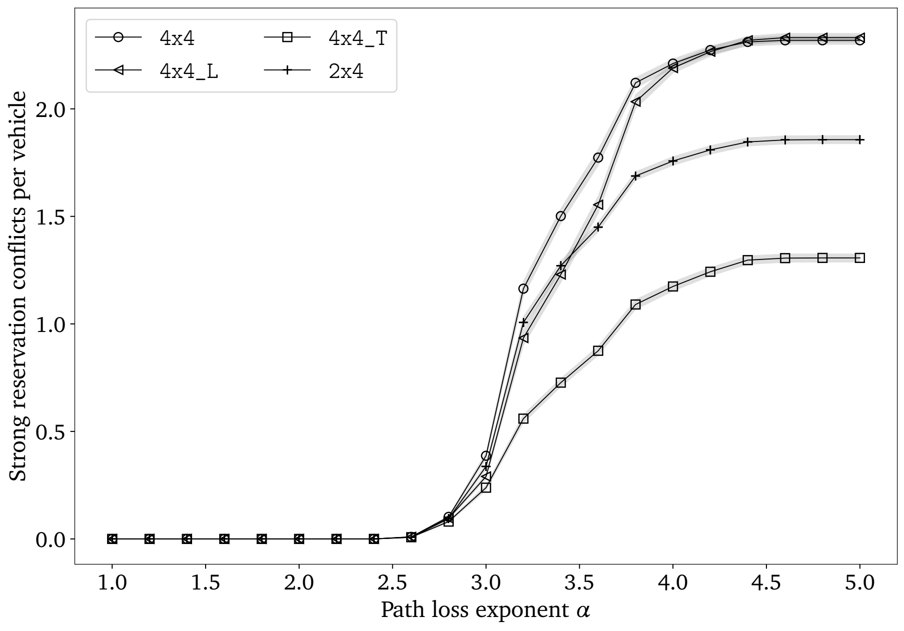


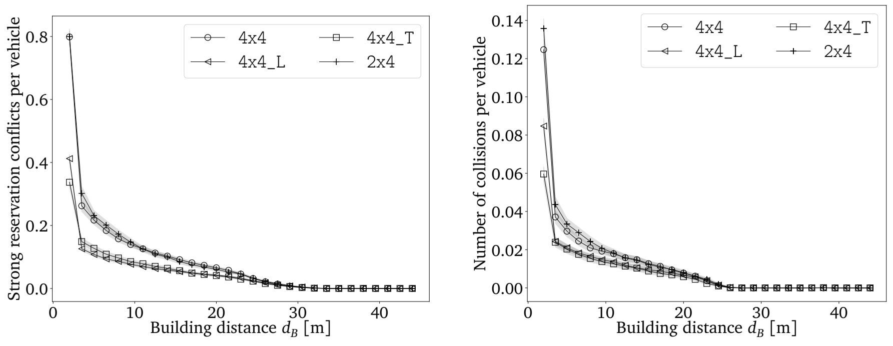
Figure 5.5 – Strong reservation conflicts per vehicle for different path loss exponents α at medium traffic demand.

To investigate its effects, I ran simulations for α values between 1 and 5 with the same remaining settings as before. As can be seen in Figure 5.5, the influence of the path loss exponent on the number of strong reservation conflicts is almost identical to that of the communication range parameter r in the unit disk model. The same applies to all other recorded metrics and traffic demand levels, which emphasizes that the path loss exponent is essentially just another way to specify a maximum communication range. Furthermore, the simulated delay seems to have no notable consequences whatsoever.

Comparing these simulation results to those of the unit disk model, one can observe that a communication range of about 80 m roughly corresponds to a path loss exponent of $\alpha = 3$ and the sufficient communication range of about 120 m is comparable to a value of $\alpha = 2.3$. Relating this to the common interpretation of the exponent yields another interesting result: If a regular, but still very simplified, free-space path loss with $\alpha = 2$ is used to model wireless signal attenuation, the resulting communication range is sufficient for the coordination protocol to ensure safe crossing maneuvers. The protocol only starts to produce collisions for values greater than about $\alpha = 2.7$, which is a suitable value to describe outdoor environments with weak attenuation effects [26]. Therefore, the controller performs reasonably well under communication conditions that are less than perfect, but it shows increasingly unsafe behavior if the effective communication range decreases further. Again, a lower d_c value or even a more sophisticated communication protocol could most likely alleviate this issue.

After having studied the effects of a limited communication range, it is interesting to investigate the situation where messages are blocked by obstacles in the line of sight between two vehicles but the communication range is effectively unlimited if the line of sight is unobstructed. When using the *obstacle shadowing* communication model of Veins in combination with obstacles at the roadside, the area in which a vehicle is able to communicate does not have a fixed, uniform shape but changes dynamically with the vehicle's position relative to the obstacles. This models the situation at urban intersections with buildings and other obstacles more realistically than the unit disk or simple path loss model. The most realistic configuration would be a combination of the two models such that messages are lost if they are blocked by obstacles or experience too high path loss. However, it is important to investigate the obstacle shadowing model alone first.

To this end, I ran simulations for the d_B parameter that specifies the distance between the buildings and the incoming and outgoing roads in the scenario as illustrated in Figure 4.2. I chose 2 m as the minimum distance to simulate the case where the buildings almost reach into the intersection, modeling the worst case scenario. Using a geometrical construction for the case where two vehicles approach the intersection on neighboring roads and at their respective maximum distance and speed (see communication range parameter), a sufficient building distance of about $d_B = 40\,\mathrm{m}$ can be determined, which is why I chose a maximum distance slightly above this value.



(a) Strong reservation conflicts

(b) Collisions

Figure 5.6 – Strong reservation conflicts and collisions per vehicle for medium traffic demand.

Figure 5.6 displays the numbers of strong reservation conflicts and collisions per vehicle for a traffic demand of $D = 500\,\mathrm{veh}/(\mathrm{h\,lane})$. Similarly to the communication range parameters, there is a value for the building distance at which the vehicles can communicate and coordinate without restrictions. In this scenario, it is at just above 30 m, which is, again, well below the theoretical upper bound. Apart from that, the plots show two more interesting results: First, increasing the building distance just slightly above the minimum value of 2 m drastically decreases the number of conflicts and collisions. The reason for this is that with a very small distance, even vehicles that are waiting at the intersection have a highly limited field of view and their communication with other vehicles on the neighboring approaches is thereby almost completely obstructed. This prevents effective coordination with those vehicles in the majority of cases, leading to a relatively large number of strong reservation conflicts. Due to the intersection geometry and building placement, however, moving the buildings a small distance further away from the intersection increases the field of view by a significant amount, as illustrated by Figure 5.7.

After this initial drop, the numbers of conflicts and collisions decrease more slowly until they reach zero at around 30 m. Taking a closer look at the curve, a distinct change of slope at about 22 m becomes obvious, which leads to the next interesting result. Moving the buildings further away from the intersection generally improves the chances of successful coordination for an increasing amount of vehicles, which explains the relatively steady, monotonic decrease of conflicts. However, the *first* vehicles that enter the simulation scenario can only coordinate effectively for large d_B values because they send their claim and reservation messages as soon as

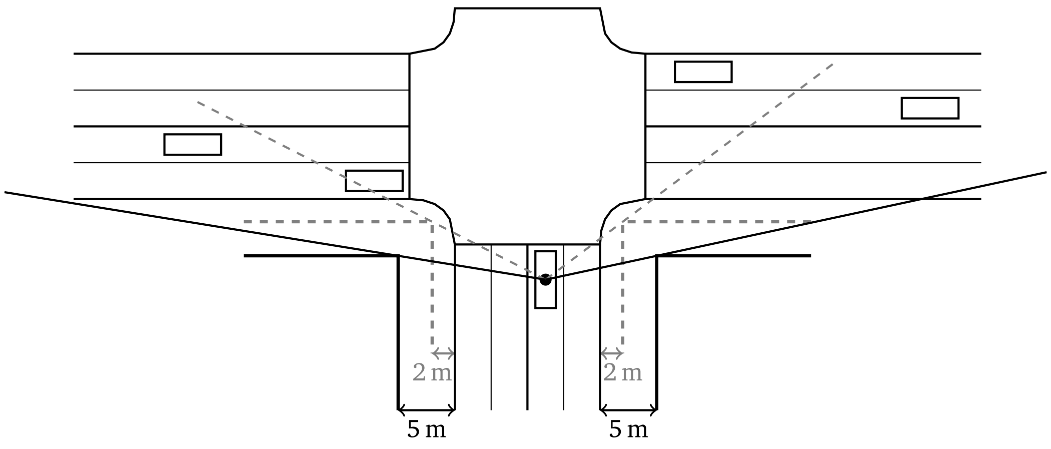


Figure 5.7 – Effect of increasing the building distance close to its minimum value. The dashed gray lines mark a building at the minimum distance of $d_B = 2\,\text{m}$ and the corresponding limits of the ego vehicle's field of view. The solid black lines indicate the same for a building that is placed 3 m further away from the roads. Even this small increase leads to a much larger field of view.

their distance to the intersection falls below the threshold value d_c. The following vehicles must wait until their leaders have entered the intersection, due to the definition of the *crossing ahead check ca*(ego). If the distance between the first and the following vehicles is small enough, the followers will start their coordination procedure at a closer distance to the intersection and, therefore, have better chances to coordinate successfully. This means that for building distance values between 5 and 22 m, the majority of the reservation conflicts and collisions is caused by the very first vehicles in the simulation. Plotting the time stamps of strong reservation conflicts that occurred across all simulation runs in a histogram confirms this, as Figure 5.8 illustrates using the 4x4 scenario as an example. As a consequence, the number of conflicts decreases more rapidly as soon as these first vehicles start to communicate effectively, which happens at a building distance value d_B of around 22 m.

Combining these two results, it becomes clear that the effects of obstacles that prevent successful communication vary greatly depending on the obstacles' positioning and the positions at which the vehicles start to send coordination messages. In general, the impact of the obstacles decreases when their distance to the intersection increases. Leaving enough room that vehicles close to the intersection have a line of sight to large parts the neighboring approaches already prevents most reservation conflicts in this configuration, but a relatively large distance is required to ensure the safety of all vehicles.

It must be mentioned here that this result strongly depends on the design decisions made for the controller implementation: The main reasons for the problematic

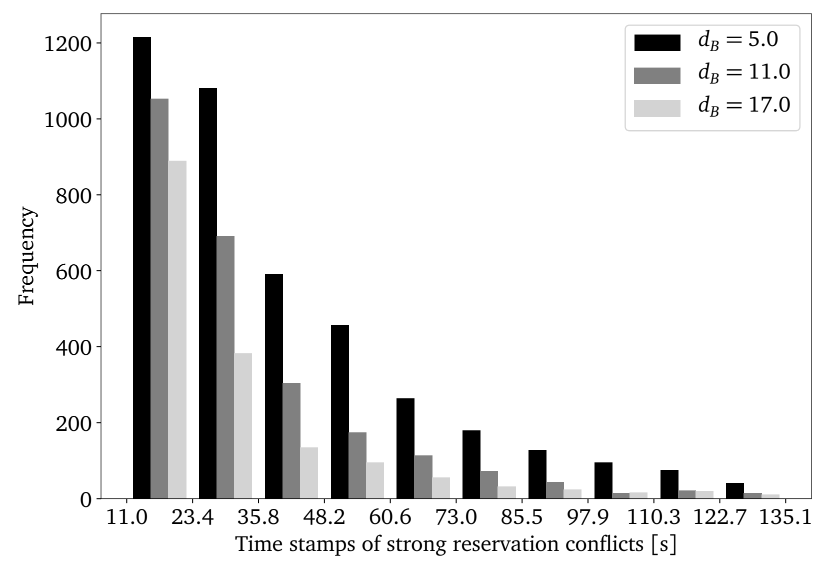
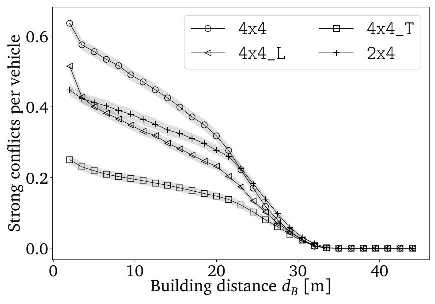


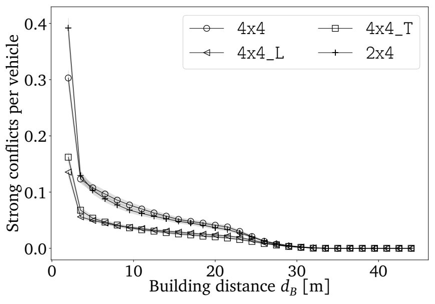
Figure 5.8 – Distribution of strong reservation conflict times in the 4x4 scenario at medium traffic demand. Each triple of bars specifies the total number of time stamps between the two enclosing simulation time values that occurred across all 200 repetitions.

coordination between the first vehicles in the simulation are the early placement of reservations and the missing possibility to update traffic information and abort the maneuver at a later point in time. Interpreting the controller automaton differently or using a more reliable technique for synchronizing traffic information could improve the situation significantly. Therefore, the relatively low level of safety in the presence of communication-blocking obstacles can be attributed to the simplified communication protocol used for this implementation and is not necessarily a weakness of the crossing controller itself.

Apart from that, the simulation results allow some additional observations that have not been discussed yet. All the above findings are based on a medium traffic demand value of $D = 500\,\mathrm{veh/(h\,lane)}$. Considering the low and high demand values as well reveals another interesting effect. As illustrated by Figure 5.9, a low traffic demand of $200\,\mathrm{veh/(h\,lane)}$ leads to a significantly smaller initial drop and a generally much larger number of reservation conflicts per vehicle, whereas the high traffic demand value of $800\,\mathrm{veh/(h\,lane)}$ produces fewer conflicts overall. This corresponds exactly to the observation that the distance between the first vehicles and their followers affects the required building distance for effective communication: If the traffic demand is low, these distances will be larger because the vehicles are inserted at a lower frequency. Conversely, a higher demand leads to smaller



(a) $D = 200\,\text{veh}/(\text{h lane})$



(b) $D = 800\,\text{veh}/(\text{h lane})$

Figure 5.9 – Strong reservation conflicts per vehicle – Comparison of low and high traffic demand.

distances and, therefore, more effective coordination and fewer conflicts. Relating the maximum metric values of about 0.8 and 0.4 for the medium and high traffic demand to the difference between the demand values 500 and $800\,\text{veh}/(\text{h lane})$ leads to rough values of $0.8 \cdot 500 = 400$ and $0.4 \cdot 800 = 320$ conflicts per hour and per lane, confirming that the lower ratio of conflicts in the high demand scenario is not only caused by the larger total number of vehicles. If these numbers were similar, it would mean that the initial phase of the simulation causes an equal number of conflicts for both traffic demands.

Furthermore, the results can be compared to the effects of a limited communication range, as introduced by the unit disk or simple path loss model. The most significant difference is that the overall number of strong reservation conflicts per vehicle is much lower for the building distance parameter than for the communication range (cf. Figure 5.5, where all scenarios exceed 1 conflict per vehicle), even at the minimum values of both parameters. This is due to the fact that in the obstacle shadowing scenario, the communication between vehicles on opposite sides of the intersection is not restricted, preventing roughly half of all reservation conflicts and collisions. The impact of this is so large because it not only prevents strong conflicts between these vehicles, but it also forces some of them to stop and wait at the intersection, which allows them to communicate better with the neighboring approaches and, additionally, it gives the following vehicles a chance to close the gap and improve their coordination as well. This effect is even more prominent in the 4x4_L scenario with the left-turning lanes because vehicles performing a turn to the left naturally have a higher potential for conflicts with the opposing traffic, causing both sides to wait more frequently than in the regular 4x4 scenario. That is why there is a significant difference between the two scenarios in the obstacle shadowing configuration but not in the unit disk and simple path loss settings. Comparing the average trip times in the 4x4 and 4x4_L scenarios confirms this, as can be seen in Figure 5.10.

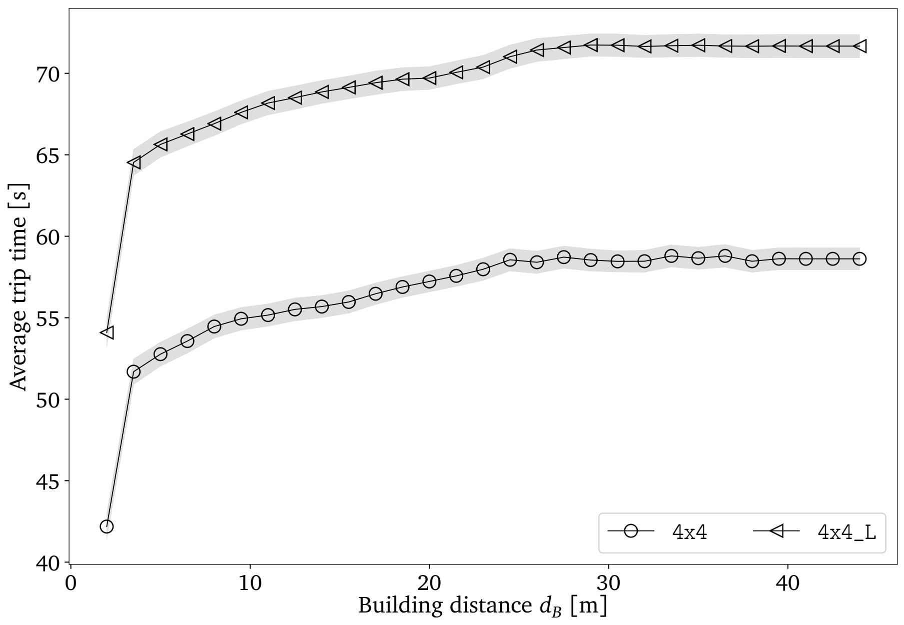


Figure 5.10 – Average trip times in the 4x4 and 4x4_L scenarios at medium traffic demand.

In contrast, the asymmetrical 2x4 scenario performs worse than before because the vehicles on the two smaller approaches with only a single lane tend to have much fewer potential conflicts with each other and, therefore, slow down less frequently. The 4x4_T variation appears to perform equally well in both the settings with limited communication range and obstacle shadowing.

This shows that in the presence of obstacles limiting the vehicles' ability to communicate based on their relative positions, the intersection structure and the traffic characteristics can have both positive and negative effects on the controller's safety performance. More generally, the purely range-based unit disk and simple path loss communication models show significant differences to the more realistic, line of sight-based obstacle shadowing model. In both cases, however, it is likely that suitable modifications of the communication protocol can solve most of the occurring coordination problems.

The communication parameters discussed so far only have deterministic effects; a message is either received by another vehicle or not, depending solely on the relative positions of the vehicles and obstacles. However, the probability for successfully receiving a wireless message in reality depends on a multitude of additional factors, many of which are far less predictable. Apart from that, it is interesting to investigate the probability for message loss itself independently to view the impact of unreliable, asynchronous traffic information from another perspective. The *artificial message loss* parameter enables this by introducing the probability p for dropping a received message as a simulation parameter.

To get a complete view on the effects of this parameter, I ran simulations for the whole range of probability values from 0 to 1. Because the parameter implementation uses the random number generator at runtime, unlike the traffic generation that applies randomness independently of simulation runs, I repeated each run with four different runtime seeds for 50 random traffic seeds.

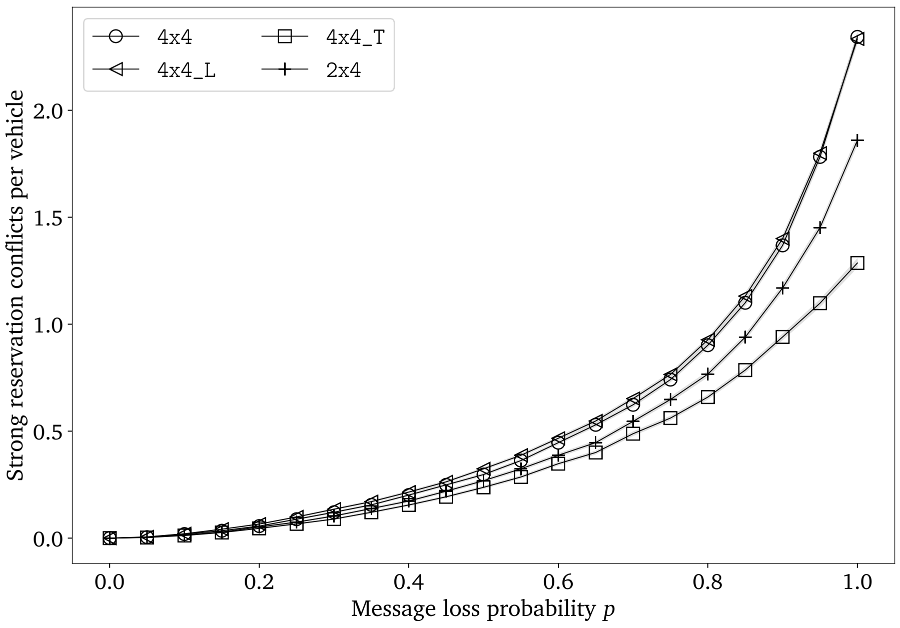


Figure 5.11 – Strong reservation conflicts per vehicle at medium traffic demand.

Figures 5.11 and 5.12 display the number of strong reservation conflicts and collisions per vehicle for a medium traffic demand value. The graphs show relatively clear exponential curves for all intersection scenarios. This may have various causes that are difficult to differentiate and analyze because of multiple interacting factors including probabilities for vehicles to execute conflicting maneuvers, the duration of each crossing maneuver, the type of messages that are lost, and the temporal offset between vehicle insertions. However, some of these factors can provide at least a basic understanding of what is happening at the intersection.

To start with, consider two vehicles A, B that plan conflicting crossing maneuvers and approach the intersection at roughly the same time. The minimal number of coordination messages that must get lost for a strong reservation conflict to occur

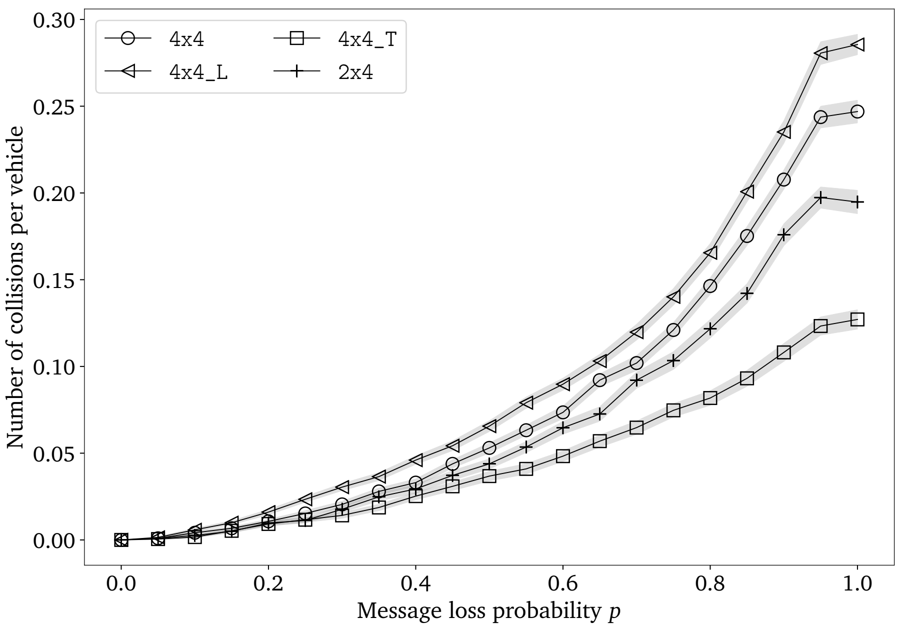


Figure 5.12 – Collisions per vehicle at medium traffic demand.

is two: It can be assumed without loss of generality that vehicle A sends its claim message `cc` first. If B receives this message, it will detect the conflict and wait until A has finished its maneuver before placing its own claim. Therefore, A's claim message is the first one that must get lost. The same logic applies to A's reservation message `rc` because A will also be the first vehicle to place its reservation and receiving the message would cause B to withdraw its claim, again preventing a conflict. Thus, at least these two messages must be lost; the probability for this is p^2.

Adding more vehicles to this scenario makes it much more complicated because it creates more possibilities for conflicting maneuvers and introduces more constraints on messages that must get lost for certain conflicts to occur. However, the probability for any specific combination of conflicting maneuvers will be a polynomial of p whose degree increases with the number of constraints. Since p is always ≤ 1, simultaneous conflicts of many vehicles are generally less likely to occur than simple conflicts of, for example, just two vehicles.

Observe that one pair of vehicles can only cause a single reservation conflict and the number of conflicts in which a specific vehicle is involved is bounded by a relatively small constant; for instance, the 4x4 scenario has only eight incoming lanes and most of its internal lanes have an even smaller number of foes. Therefore, the number of reservation conflicts per vehicle can only exceed 0.5 if many vehicles are involved in more than one conflict. A possible cause for this could be chains of conflicts, where the temporal offset between vehicles A, B is relatively large and as soon as A finishes its maneuver, another vehicle C has a conflict with B, followed by a conflict of C and D after B has left the intersection, and so on. Figure 5.11 shows that the number of conflicts per vehicle only exceeds 0.5 for values of $p \geq 0.6$, which supports these considerations because losing more than half of all messages already causes extremely unreliable coordination information.

Finally, the number of possible conflict combinations for a set of n vehicles is roughly equal to the number of subsets of a set with n elements that have a size ≥ 2, which is $2^n - n - 1$, and the number of such subsets of size exactly $k \leq n$ is the binomial coefficient $\binom{n}{k}$. Both of these terms grow rapidly with increasing n. Although these are only rough approximations of the real number of reservation conflicts that can possibly occur, they indicate that there is a huge potential for different kinds of conflicts that may occur if the coordination of the vehicles is unreliable enough. All of these considerations are major factors causing the exponential behavior displayed by the graphs.

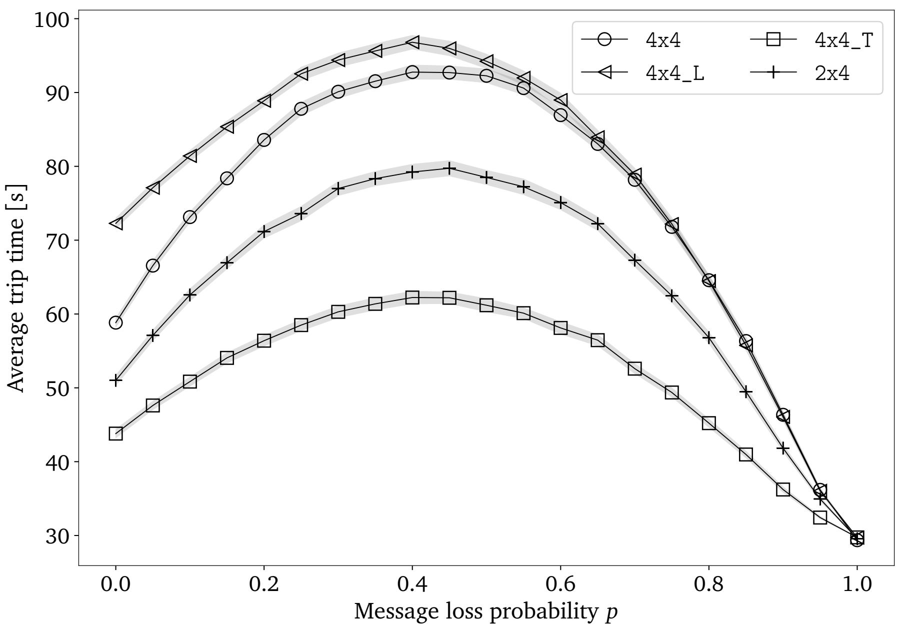


Figure 5.13 – Average trip times at medium traffic demand.

Moving on, there is an interesting artifact in Figure 5.12 that also needs to be discussed: The number of collisions per vehicle does not continue on its exponential trajectory until the maximum value of $p = 1$, but shows a much slower increase, or even a slight decrease, after about $p = 0.9$. This behavior can be explained by considering the extreme situation at $p = 1$: Here, no coordination messages are received and the vehicles are thereby almost completely unaware of each other. Each vehicle only keeps its safety distance to the leading vehicle and waits until it has entered the intersection until it starts its own crossing maneuver. Therefore, the vehicles rarely slow down and perform their maneuvers at the maximum possible speed. This is clearly visible when plotting the average trip times, as can be seen in Figure 5.13, where the average trip times in all scenarios reach a global minimum at $p = 1$. As a consequence, the vehicles spend relatively little time in the conflict zones of the intersection.

Changing the situation by decreasing the probability for losing messages to 0.95 causes a small fraction of the vehicles to wait after receiving a claim or reservation message. Those vehicles will then have to accelerate again for their crossing maneuver, increasing the time spent on the intersection. This also slows down all following vehicles and thereby increases the density of vehicles at the intersection. In turn, the probability for colliding with one of the vehicles that did not wait increases for at least the first of the waiting vehicles. Looking at the squared relative velocity of the collisions confirms this: As illustrated by Figure 5.14, the squared relative collision velocity differs significantly between $p = 0.9$ and $p = 1$. Decreasing the message loss probability further, however, suffices to make the coordination more effective and leads to fewer conflicts and collisions.

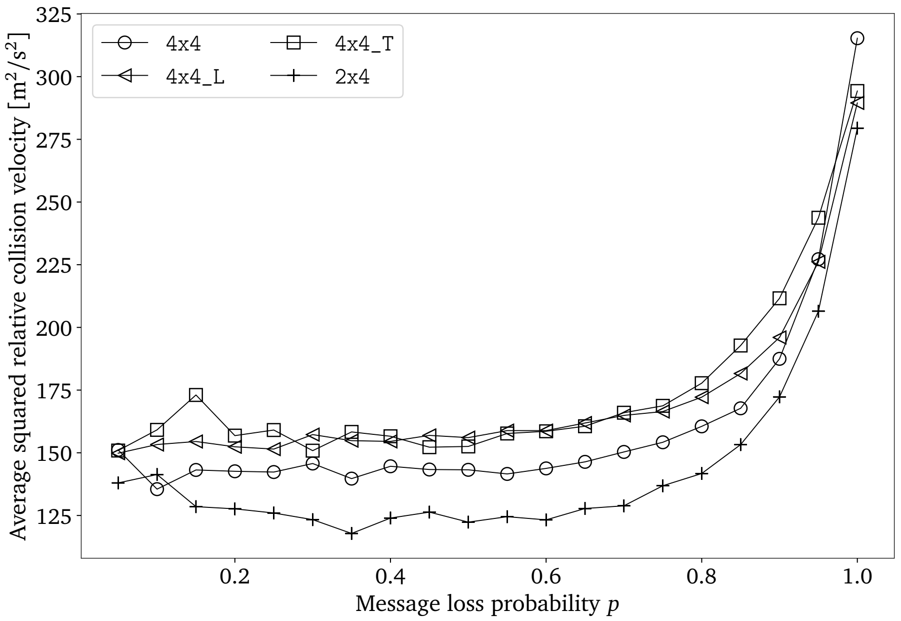


Figure 5.14 – Average squared relative collision velocity at medium traffic demand. Note that there are no collisions for $p = 1$.

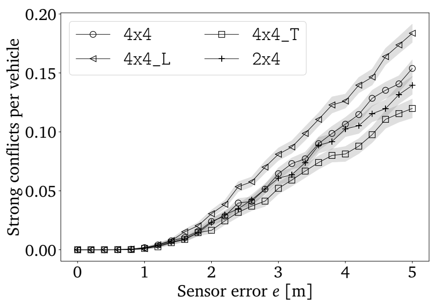
To conclude this discussion, it can be said that the loss of coordination messages is not quite as problematic as could be expected, but causes a tremendous risk of collisions if very few messages are received. A loss probability below 0.2 only causes a small number of reservation conflicts and very few collisions. At this point, message loss has a greater influence on the average trip time than on the safety of the vehicles. However, for larger values, its effects can become much more dramatic because a malfunctioning coordination protocol can lead to many vehicles starting conflicting maneuvers at the same time. Viewing this from the opposite side, though, receiving just 20 % of all messages already reduces the risk for a collision significantly. In reality, sophisticated communication protocols can easily avoid most loss of information caused by unpredictable effects like random loss of messages.

Comparing this parameter to the previously discussed communication models and their parameters, the most obvious difference is that the coordination only provides perfect safety for the minimum message loss probability $p = 0$. All previous parameters have a threshold value such that the protocol works without problems for all values beyond the threshold. This has to do with the direct nature and the independence of the message loss parameter: Because every single coordination message is potentially affected, there is nothing to limit the influence of the parameter on the effectiveness of the communication. For example, vehicles waiting at the intersection previously had better chances at communicating successfully; the message loss parameter is completely independent of the vehicles' positions and affects them all the same.

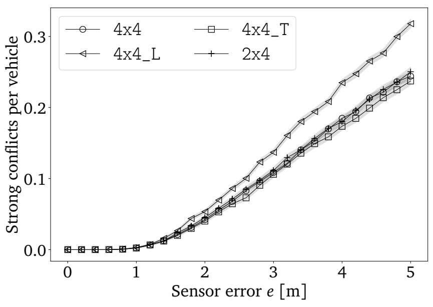
Apart from the communication models and parameters that directly influence the vehicles' ability to communicate, there are two more simulation parameters with different effects. To start with, the *sensor error* parameter e introduces random errors to the perception of other vehicles' positions which are used in the *potential collision check pc* to determine whether or not it is safe to place or keep a claim. The value e specifies the standard deviation of the normally distributed error value that is determined randomly and added to the perceived position of another vehicle every time this position is requested by the ego vehicle. This mimics random noise in the output of the vehicle's on-board sensors.

I conducted simulations for error values between 0 m and 5 m to also include unrealistically large errors. Similar to the message loss parameter, which also generates random numbers at runtime, I used 50 random traffic seeds and repeated each run four times with different runtime seeds.

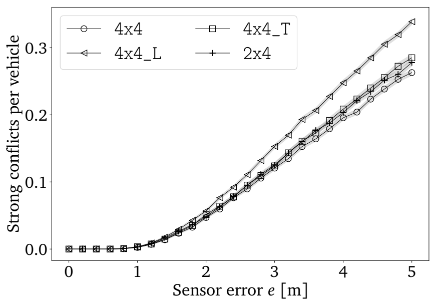
As depicted by Figure 5.15, the number of strong reservation conflicts per vehicle increases linearly with e for values of $e \geq 1$ in all scenarios and for low, medium, and high traffic demand levels. For higher traffic demand, the number of conflicts is generally larger; this can be explained by the higher density of vehicles that causes crossing maneuvers to be executed more frequently. A more interesting result is that across all simulation runs, only a negligible number of collisions occurred, and only for large parameter values of $e > 3$. Especially compared to the previous parameters, this means that strong reservation conflicts have an extremely low risk of actually



(a) $D = 200\,\text{veh}/(\text{h lane})$



(b) $D = 500\,\text{veh}/(\text{h lane})$

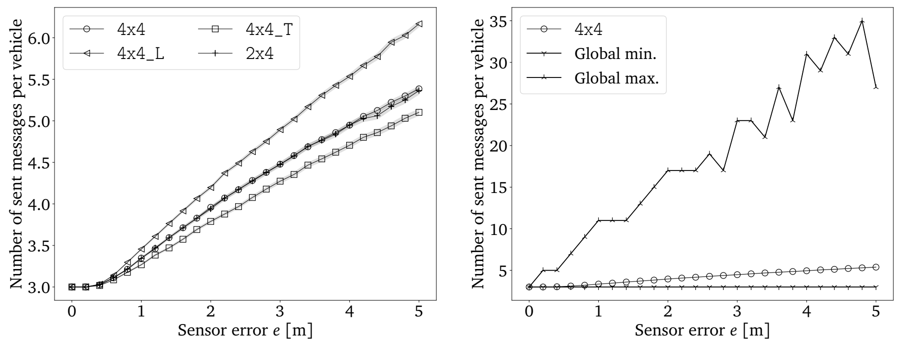


(c) $D = 800\,\text{veh}/(\text{h lane})$

Figure 5.15 – Strong reservation conflicts per vehicle for low, medium, and high traffic demand.

leading to collisions. The reason for this is that the circumstances that could allow a collision are far less likely to occur than strong reservation conflicts in general: Two vehicles must place their claims and reservations on conflicting internal lanes when their distances to the intersection are roughly equal. This means that at least one of these vehicles must have positive sensor errors large enough that the other vehicle appears to have already passed its clearance position. Moreover, such an error must occur in at least two time steps because the *potential collision check* is performed several times before a reservation is eventually placed. On the other hand, a strong reservation conflict can also occur when one of the vehicles is already very close to its clearance position, in which case only a small sensor error is required. A collision, however, is almost impossible at this point because this vehicle will have left the conflict zone before it is reached by the second vehicle. Since the sensor error is normally distributed, the risk of collisions is extremely small, even though strong reservation conflicts occur frequently.

Furthermore, the introduced errors are centered around zero, which makes negative errors that move the perceived vehicle position backwards just as likely as positive errors. This can cause the opposite effect of vehicles detecting a potential collision although the vehicle on the foe lane has already passed its clearance position.



(a) Comparison of intersection scenarios **(b)** Details of 4x4 scenario

Figure 5.16 – Average number of sent coordination messages per vehicle. The global minimum and maximum in Figure 5.16b specify the minimum and maximum number that occurred across all vehicles in all repetitions.

One consequence of this is that vehicles generally have to send more coordination messages, as illustrated by Figure 5.16. The exact process that causes this behavior only requires a vehicle to perceive its desired internal lane as free for one time step and to detect a potential collision in the next step, so that it sends a claim message only to withdraw the claim one trigger interval later. Each of these events can be caused by small random errors when all foe vehicles are close to their clearance positions. They also do not have to occur at a specific time because the *potential collision check* is performed in each time step as long as the reservation is not placed. Together, this explains why the number of sent messages already increases for values of $e < 1$, that is, before strong reservation conflicts occur.

Intuitively, one might think that this behavior causes vehicles to slow down and wait more often because it takes more coordination messages to finally negotiate a maneuver and place the reservation. As Figure 5.17 shows, however, the average trip times do not increase at all; in fact, they even decrease slightly. This is the case because on average, the vehicles are still able to place their reservation quickly enough that there is no need to slow down or the trip time is not affected significantly. Looking at Figure 5.16 again, the average number of sent messages is relatively small: Every vehicle must send at least three messages (`cc`, `rc`, and `wd_rc`), and the number of messages increases by two (`wd_cc` and `cc` again) every time a placed claim must be withdrawn due to the event described above. Because the vehicle controller sends one of these messages and performs the corresponding action every time it is triggered, even withdrawing a claim three times only adds six trigger intervals to the total time until the reservation is placed, which correspond to 0.6 s. As this effect is relatively small, and since the trip times of vehicles involved in a

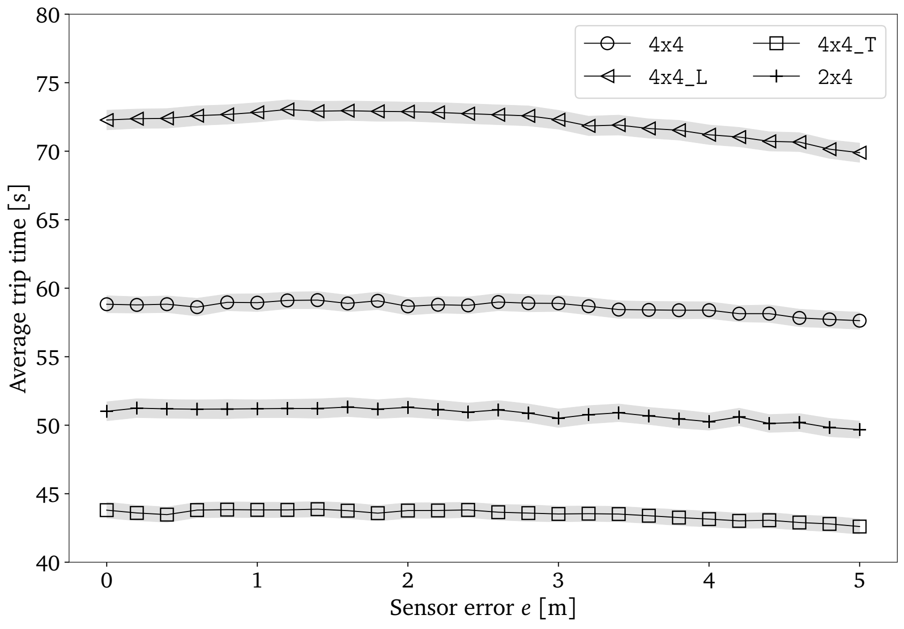


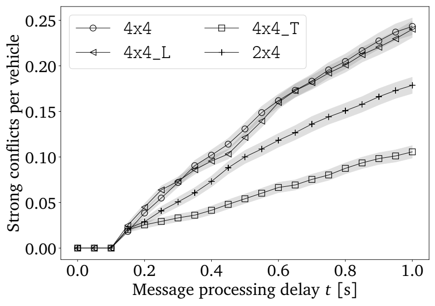
Figure 5.17 – Average trip times at medium traffic demand.

strong reservation conflict are shorter than they would be without the conflict, the average trip time does not increase.

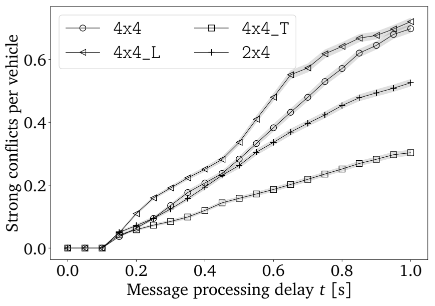
This is a remarkable difference to the previous parameters, where information was lost completely instead of just altered by random errors: For example, a message loss probability of 0.5 caused the average trip time to increase tremendously because vehicles would frequently miss withdrawal messages and have to wait until their internal information could be safely deleted, which means a delay of at least $t_c = 5\,\text{s}$ or $t_{cr} = 7.5\,\text{s}$, depending on the type of the lost message. With the sensor error parameter, however, the internal information is always up to date and only the additionally required sensor information is sometimes unreliable. While the former significantly increases both the trip time and the risk of collisions, the latter only leads to effectively harmless violations of the coordination protocol. An extremely large error value is required here to cause any collisions at all. The only noticeable effect of the sensor error that negatively affects the controller's performance is the increased communication effort, which could be the cause of further problems in a more realistic scenario or in combination with other parameters.

Finally, the last parameter to be discussed individually is the *processing delay*. Its value t specifies a delay between the time a coordination message is received and the time it is processed, i.e., the receiving vehicle's internal traffic information is updated. It mimics the combined effects of communication overhead that could be caused by more sophisticated protocols and the computation time required to interpret and process a message's content.

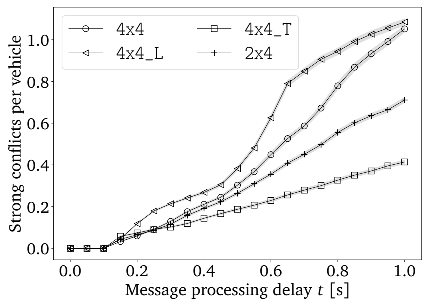
Because in reality, this delay value is unlikely to regularly exceed 1 s, I ran simulations for values between 0 s and 1 s. Each run was repeated for 100 different random traffic seeds and 2 runtime seeds.



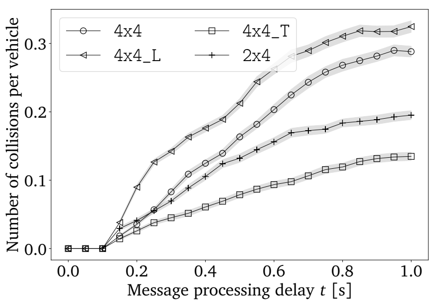
(a) $D = 200\,\text{veh}/(\text{h lane})$



(b) $D = 500\,\text{veh}/(\text{h lane})$



(c) $D = 800\,\text{veh}/(\text{h lane})$



(d) Collisions for $D = 500\,\text{veh}/(\text{h lane})$

Figure 5.18 – Strong reservation conflicts and collisions per vehicle – Comparison of low, medium, and high traffic demand.

As can be seen in Figure 5.18, the number of strong reservation conflicts per vehicle increases with the processing delay and also with the traffic demand level. The same is true for the number of collisions, which shows a relatively high risk per reservation conflict. However, no conflicts occur for delay values $t \leq 0.1$ s. It is no coincidence that this is exactly the controller trigger interval, as the sequence diagram in Figure 5.19 illustrates. It shows an exemplary communication between two vehicles A, B with conflicting maneuvers, where A sends its claim message first. Due to the processing delay of t_1 or t_2, B is unaware of this claim the next time its controller is triggered and it places its own claim, sending the corresponding message. This message has no effect on A because A's claim was placed first, giving it higher priority. Depending on the delay value, B will either withdraw its claim or not: For a relatively small value like t_1, A's `cc` message is processed before the next trigger of B, causing an immediate withdrawal. If, however, t is large enough that B

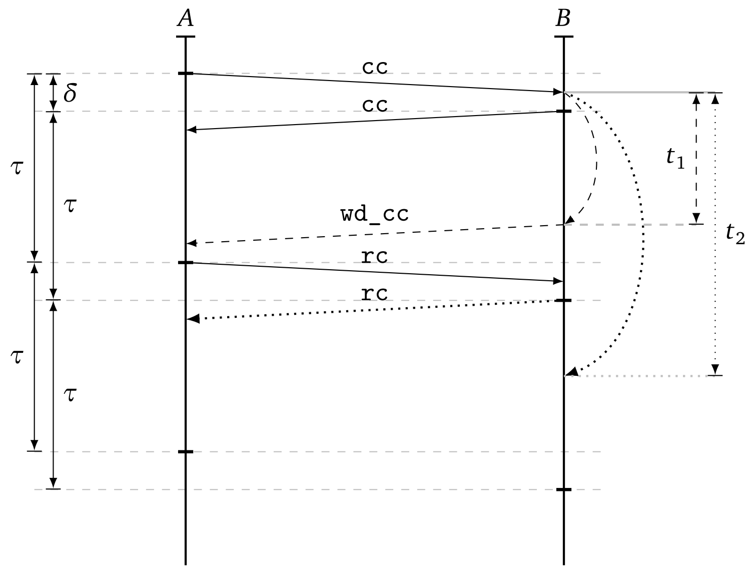


Figure 5.19 – Communication sequence diagram of vehicles A, B. Controller triggers are marked by the thick, horizontal lines on the two vertical axes. τ denotes the trigger interval and δ is the offset between the two vehicles' controllers. t_1 is a relatively small processing delay, indicated by the dashed, black lines, that is smaller than τ and causes no serious problems. t_2, indicated by dotted, black lines, is an alternative, larger delay that is greater than τ and can cause strong reservation conflicts. Only the processing time of the first `cc` message is drawn. The propagation delay of messages is only drawn for completeness and is exaggerated to be visible (without a communication model, it is 0 s).

is triggered a second time before the message is processed, e.g. $t = t_2$, B will place its reservation, leading to a strong conflict since A places its own reservation either way.

This can only occur if the time between receiving and processing a message can include two controller triggers, i.e., if $t \geq \tau$. If this does not hold, the worst-case effect is the withdrawal of a claim, as the example t_1 shows. Note that in case the trigger of B happened before the claim message of A is received, the vehicles would simply switch their roles due to the negligible propagation delay. As Figure 5.20 clearly shows, increasing the trigger interval also increases the tolerance for larger processing delay values. This emphasizes the relation between the two parameters. Naturally, the probability for strong reservation conflicts increases with the processing delay because events as illustrated for t_2 in Figure 5.19 become less restricted in the time at which the first claim message is sent.

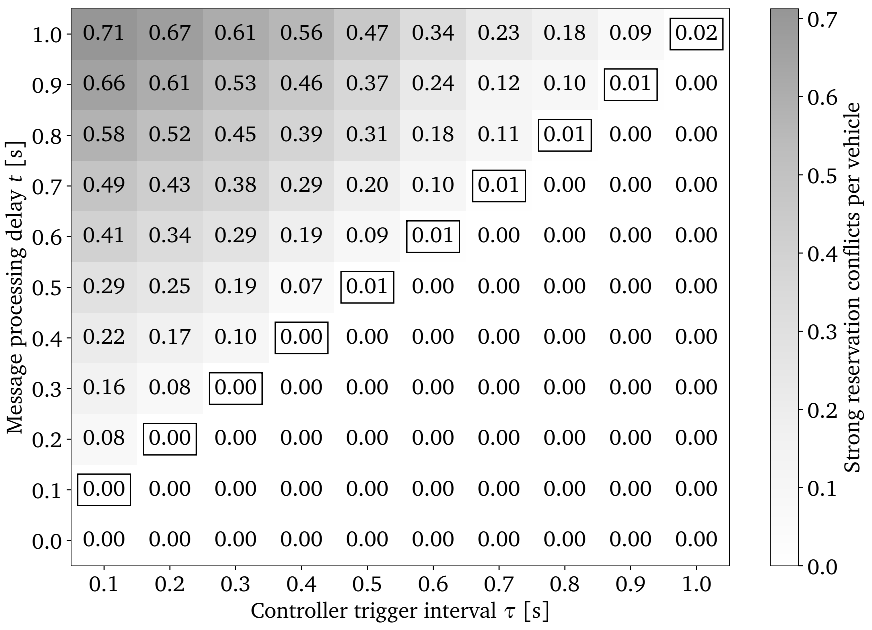


Figure 5.20 – Strong reservation conflicts for different combinations of trigger interval τ and processing delay t at medium traffic demand. The diagonal cells in which t equals τ are marked with boxes. They clearly indicate that strong reservation conflicts only occur if $t \geq \tau$ holds.

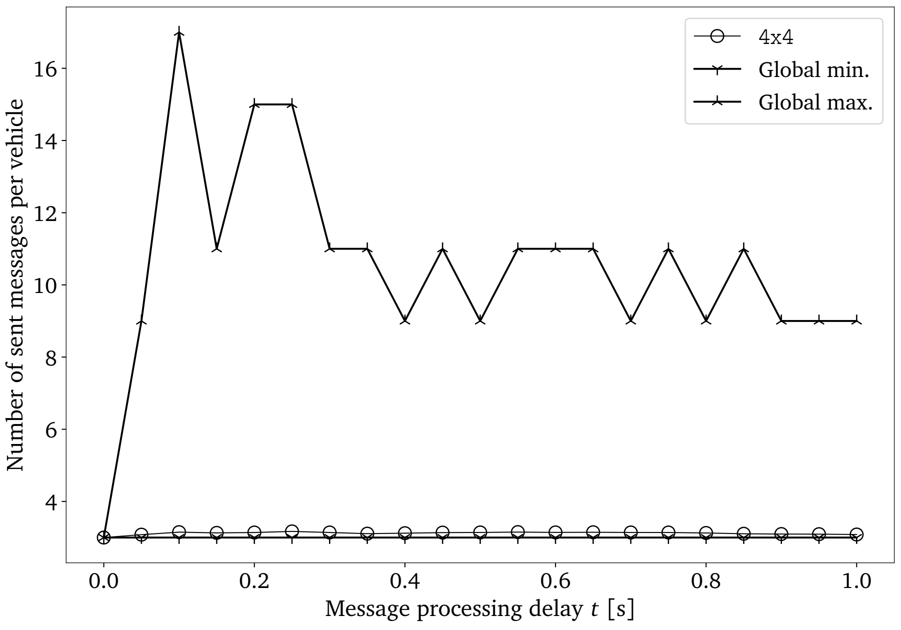


Figure 5.21 – Average and min./max. number of sent coordination messages per vehicle at medium traffic demand.

The other effect, caused by delay values like t_1 in the sequence diagram, has an influence on the number of sent coordination messages per vehicle. As Figure 5.21 displays, the average number only increases marginally, but a small amount of vehicles requires significantly more messages than the minimum value of three. It also shows that the communication is already affected for delay values $t \leq 0.1$, unlike the strong reservation conflicts. The number of messages peaks at exactly $t = 0.1$, which is the largest value at which strong conflicts are still extremely unlikely. For larger delays, the conflicts start to make the traffic situation more chaotic and the relatively tight conditions for the processing delay causing only the withdrawal of a claim are met less frequently.

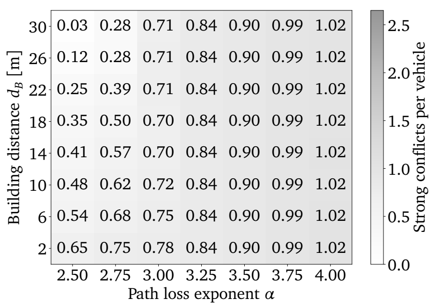
Concluding the discussion of this parameter, it can be said that the controller is actually quite resistant to processing delay because the delay value must exceed a directly controllable threshold in order to have serious, negative effects. However, if this threshold is exceeded frequently, the effects pose a significant threat to the safety of crossing maneuvers. Additionally, even delay values below the threshold can have negative effects on the efficiency of the communication because they can lead to vehicles placing claims prematurely and having to withdraw the claim immediately afterwards. Again, it is important to note that these problems can be attributed to the communication protocol and certain implementation decisions that are not forced by the abstract controller definition.

5.2.2 Parameter Combinations

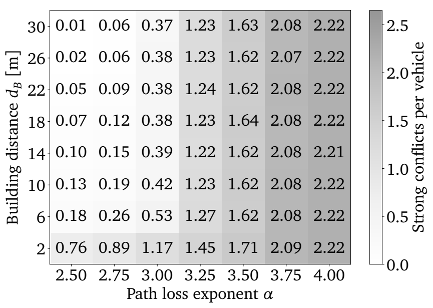
Having discussed each simulation parameter individually, the next logical step is to investigate combinations of multiple parameters. The motivation for this is that in reality, of course, the effects simulated by the parameters occur simultaneously, and there might be interactions between them which could amplify or weaken their impact on the controller's safety. It is not possible to select any particular parameter configuration as representative of the most realistic setting possible because there are no reference values, the simulation setting is still very simplified, and the conditions in reality are highly diverse. However, it is likely that somewhat realistic parameter values lie between the thresholds where the controller works without any problems and where the coordination breaks completely. It is also not feasible to simulate all possible combinations of the previously discussed parameter values. Therefore, in this section, I will focus on the most interesting combinations, introducing a number of constraints to limit the parameter space and working towards a reasonably realistic scenario.

Because the most significant differences between the intersection scenarios have already been discussed, only the 4x4 scenario is considered here. The message loss parameter was introduced to study the effects of randomly losing information independently. This is an artificial effect and in reality, message loss usually has clearly identifiable causes and is not purely random. In order to compensate for some of the effects that are not covered by the communication model parameters, the message loss probability is fixed to $p = 0.005$ for all following simulations. It is not studied any further because the other parameters are better suited for modeling a realistic scenario. Due to the greater variability of the simulation results caused by using more parameters, each simulation was repeated for 100 random traffic seeds and 4 runtime seeds, leading to a total of 400 repetitions per run to achieve statistical significance.

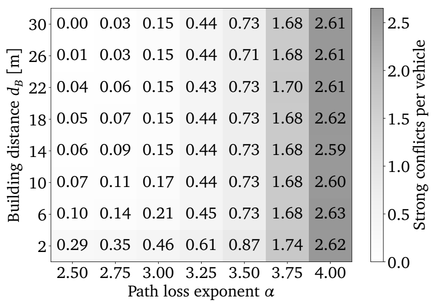
To start with, the simple path loss and obstacle shadowing communication models have the most severe impact on the safety at moderate parameter values that are not too unrealistic. For example, a distance of 20 m between the roads and the closest building or a path loss approximated by an exponent of $\alpha = 3$ might occur in a real traffic situation and can already lead to strong reservation conflicts and collisions. In contrast, position sensor errors of more than 1 m and a regular processing delay above 0.1 s can be considered to be exceptions.



(a) $D = 200\,\text{veh/(h lane)}$



(b) $D = 500\,\text{veh/(h lane)}$



(c) $D = 800\,\text{veh/(h lane)}$

Figure 5.22 – Strong reservation conflicts per vehicle for combinations of the path loss exponent α and the building distance d_B.

Both communication models simulate limitations of the area in which a vehicle can communicate with other vehicles. In reality, this area is affected both by obstacles in the line of sight and by the distance between the sending and the receiving vehicle, so it is interesting to investigate whether one of these effects dominates the other and if there are interactions that affect the controller's performance. To this end, I ran simulations for combinations of the most relevant parameter values and for low, medium, and high traffic demand, as depicted in Figure 5.22.

The plots show distinct differences between the columns, while the values inside one column show very little variation. Effects of the building distance are only visible in the first columns, i.e., in situations where the communication range is relatively large. This clearly shows that the communication range is the dominant factor. The reason for this is quite intuitive: By interpreting the obstacle shadowing model as a more complicated, dynamic form of range limitation, it becomes clear that the range at which a vehicle is able to communicate with other vehicles in any given direction is either limited by an obstacle or by the maximum range implied by the path loss, depending on which one is more restrictive. Because the buildings next to the intersection never restrict the communication between vehicles on opposite approaches, the path loss even has an impact if the smallest building distance is used. For sufficiently large α values, this also prevents many vehicles from braking and waiting at the intersection; recall that with the obstacle shadowing model, vehicles on opposite approaches often cause each other to wait, slowing down all following vehicles and thereby reducing their required communication area. This is the main reason why the simple path loss model is dominant. The effect that vehicles closely following their leaders have better chances to coordinate, however, is common to both models. That is why an increasing traffic demand still improves the overall situation by causing a higher density of vehicles, as Figure 5.22c illustrates. Because the extreme values of α practically prevent communication at any range, their effect is also extreme and not influenced by the obstacle shadowing model at all.

The remaining parameters are the sensor error and processing delay. Both of them have a threshold value at which strong reservation conflicts occur but they also influence the communication efficiency before these values are reached. In order to investigate interactions between these parameters and the communication models, they are discussed separately before combining them into a configuration that features all parameters. Because their threshold values are relatively high compared to what could be expected in a realistic scenario, only a small part of their previous range is considered. In order to prevent an explosion of the parameter space, the remaining parameters are restricted as follows: Only the medium traffic demand of $D = 500$ veh/(h lane) is used; recall that the effects of both the sensor error and the processing delay parameter increase with the demand value but have no notable further interactions. Additionally, the communication model parameters are limited to three values each. These values are $\alpha \in \{2.5, 3.0, 3.5\}$ for the simple path loss model and $d_B \in \{2, 16, 30\}$ for the building distance. They are selected such that each model has a low, a medium, and a high value which capture the most relevant impact levels and do not lie completely outside of a realistic range.

Starting with the sensor error parameter, I ran simulations for values between $e = 0$ and $e = 2$ in order to include values beyond the threshold $e = 1$ at which strong reservation conflicts occur. Figure 5.23 displays the resulting numbers of strong conflicts per vehicle for all nine combinations of the selected values for α and d_B.

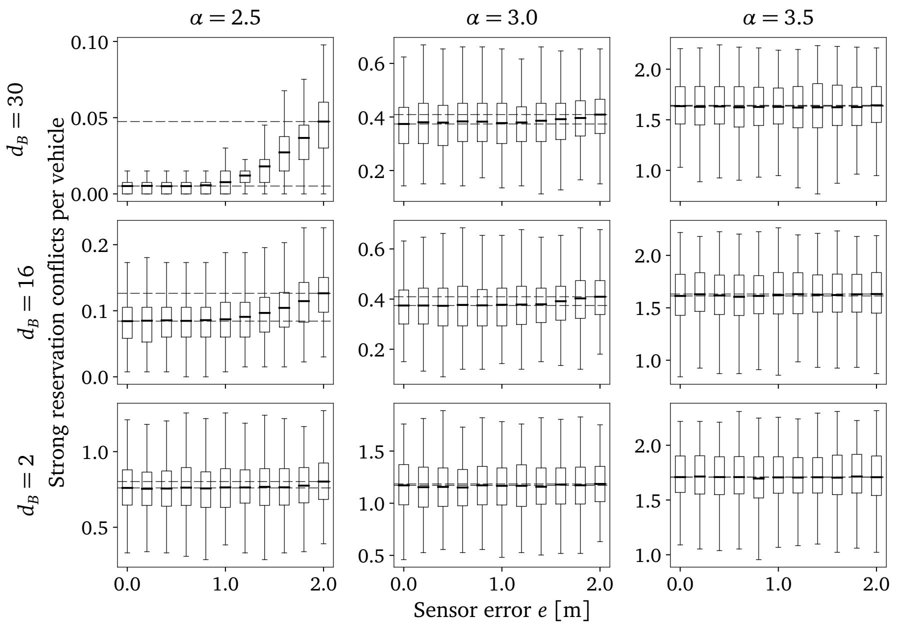


Figure 5.23 – Strong reservation conflicts per vehicle for nine combinations of the path loss exponent α and the building distance d_B, and several different sensor errors e. The simulation results for each of the nine combinations are presented as box plots. Each box has a horizontal line specifying the mean value and the box itself extends to the first and third quartiles. The whiskers extend to the most distant data points that are not further than 1.5 times the inter-quartile range away from the box. The dashed, horizontal lines depict the mean values of $e = 0$ and $e = 2$ for easier comparison.

The plots show that for most of the combinations, the additional sensor error does not have any notable effect. Its characteristic behavior that was discussed previously is most visible for the combination of $\alpha = 2.5$ and $d_B = 30$, where the communication models have the lowest impact. As their influence on the number of conflicts increases, the effects of the sensor error vanish almost completely. Moreover, in the individual discussion of this parameter, the simulation results showed that at an error value of $e = 2$ and at medium traffic demand, it leads to roughly 0.05 strong reservation conflicts per vehicle in the 4x4 scenario (cf. Figure 5.15b), which is confirmed again by the combination $\alpha = 2.5, d_B = 30$. Although this is a relatively small number, it should make a visible difference if the sensor error's effects are still present for the other combinations. However, the horizontal lines in Figure 5.23 and the computed differences both show that the total difference made by the sensor error becomes negligible for combinations where at least one of the communication model parameters is at its highest value. This indicates that the effects of the sensor error parameter are not only shadowed by the larger impact of the communication models but many reservation conflicts that could be caused by sensor errors are actually prevented, either because the same conflicts occur earlier due to insufficient communication possibilities or because the more chaotic traffic dynamics produce fewer situations susceptible to sensor errors.

Furthermore, the simulation results also show that the sensor error has no impact on the number of collisions per vehicle whatsoever. This is not surprising because the sensor error alone already did not cause collisions, but it confirms that there are no interactions with the communication models.

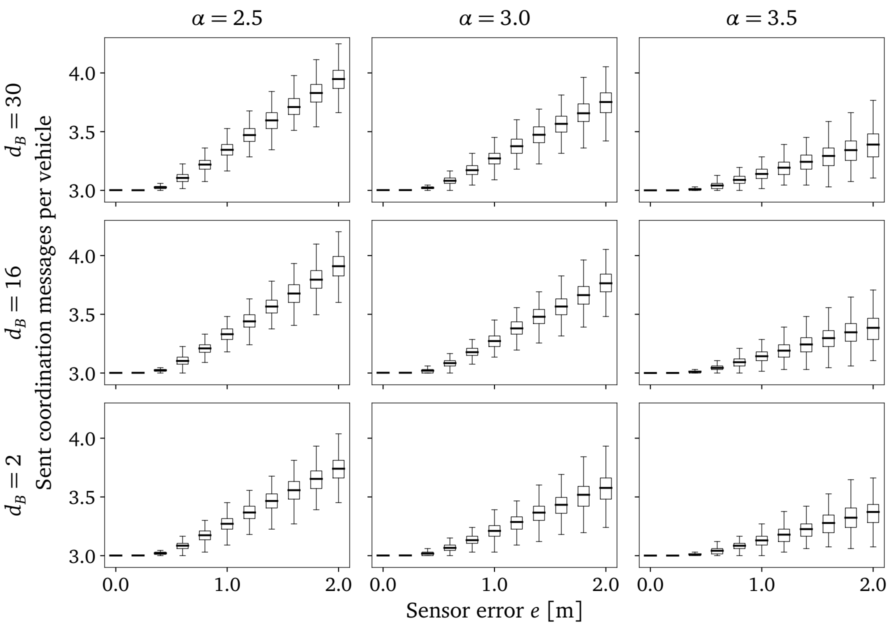


Figure 5.24 – Average number of sent coordination messages per vehicle in the combined sensor error simulation.

Apart from reservation conflicts and collisions, the other effect of the sensor error parameter that was discussed previously was the increased number of coordination messages sent by each vehicle. Figure 5.24 clearly shows that the parameter has the same effect for all combinations of α and d_B, but increasing either of the communication model parameters significantly reduces its magnitude. The reason for this is that sensor errors can only cause the ego vehicle to withdraw a claim if it is aware of the reservation of another vehicle; otherwise it will not even request positional information. The communication models can prevent the vehicle from receiving `rc` messages, leading to much fewer situations in which vehicles have to withdraw their claim and, as a consequence, reducing the number of sent messages.

In summary, the sensor error parameter is dominated by both communication models although it is only indirectly related to the communication itself. Its effects generally become less significant for larger values of α and d_B, but the strong reservation conflicts caused by sensor errors vanish completely, whereas the number of sent coordination messages is still affected to some extent for all investigated combinations.

I conducted a similar simulation for the processing delay parameter, using values between $t = 0$ and $t = 0.2$. Figure 5.25 shows the average number of collisions per vehicle for all communication model parameter combinations.

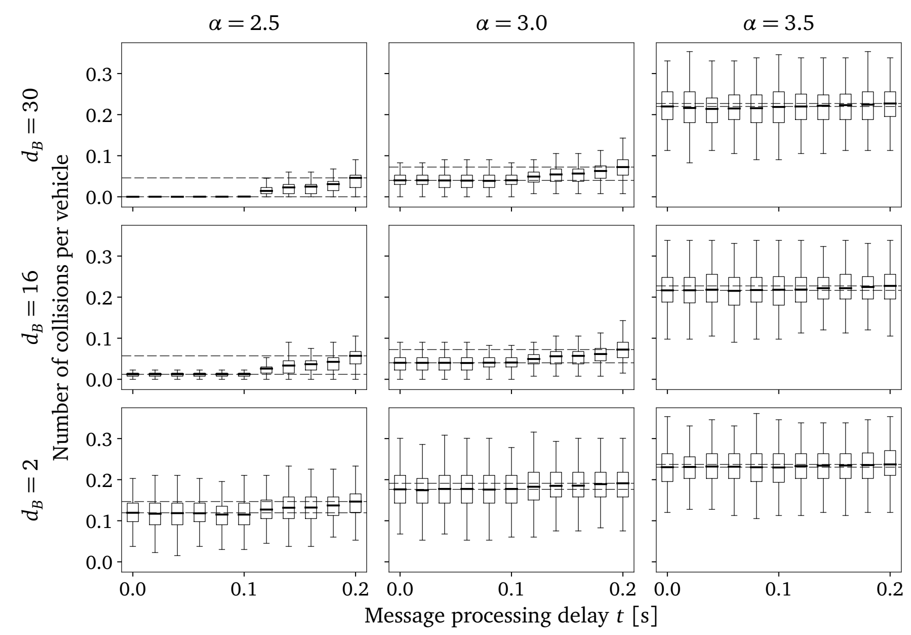


Figure 5.25 – Average number of collisions per vehicle in the combined processing delay simulation.

The simulation results are very similar to the results for the sensor error parameter, apart from the fact that the processing delay also affects the number of collisions; this was the major difference between the parameters in their individual discussions. In addition to that, the processing delay parameter generally shows a more significant impact on the safety for all parameter combinations so that its effects are even noticeable for the combination $\alpha = 3.5, d_B = 30$ where the communication models completely dominate the sensor error. Conversely, the processing delay affects the number of sent messages slightly less than the sensor error parameter. These results make sense because they perfectly reflect the findings of discussing each parameter individually: While the sensor error significantly increases the number of sent coordination messages and only causes strong reservation conflicts without collisions, the processing delay parameter leads to reservation conflicts with a high risk for collisions and increases the number of coordination messages only marginally. Both parameters are dominated by the communication models because their effects can only cause reservation conflicts under relatively restricted conditions which are met less frequently if the communication possibilities are too limited.

Having discussed these two parameters in combination with the communication models, the final step is combining all parameters in one simulation. To this end, I limited the granularity of both the sensor error and processing delay parameters but used the same value ranges as before, i.e., $e = 0$ to $e = 2$ and $t = 0$ to $t = 0.2$, respectively. Figures 5.26 and 5.27 display the simulation results for the number of strong reservation conflicts and sent coordination messages per vehicle.

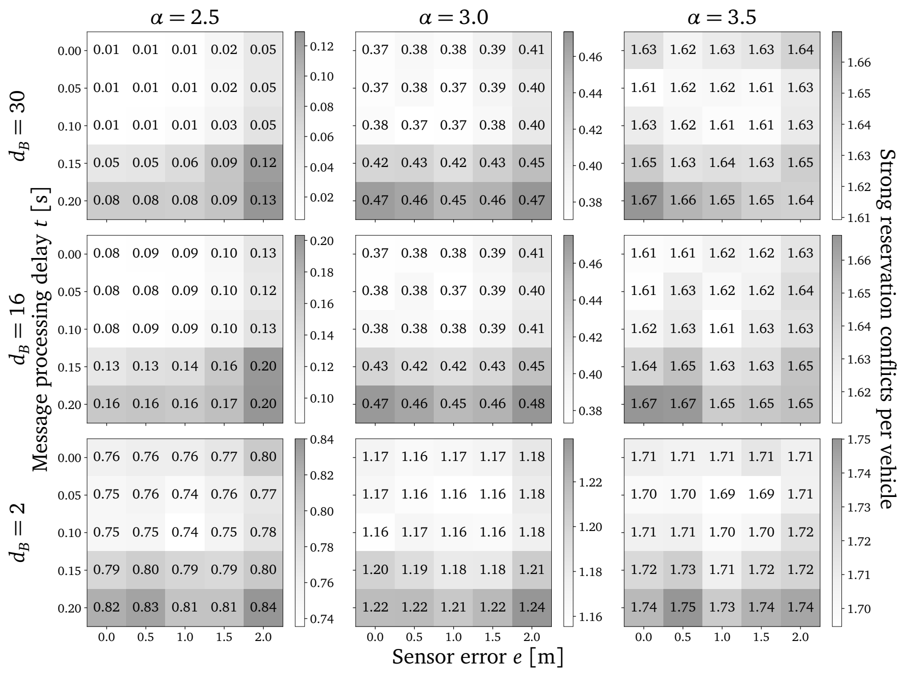


Figure 5.26 – Strong reservation conflicts per vehicle in the combined sensor error and processing delay simulation.

For the smaller values of α and d_B, the plots show the same results as discussed above, namely that the processing delay parameter has a stronger impact on the safety, i.e., strong reservation conflicts, and the sensor error has greater influence on the number of sent messages, while both are dominated by the communication model parameters. For the strong reservation conflicts in Figure 5.26, the processing delay threshold at a value of $t = 0.1$ is clearly visible in all combinations. However, the threshold $e = 1$ of the sensor error parameter does not stand out as much. For Figure 5.27, the opposite is true. The processing delay and sensor error parameters relate to each other in a different way than the simple path loss and obstacle shadowing communication models: There are no obvious interactions that amplify or weaken their combined effects and one does not dominate the other, but they complement each other in the sense that if one of them is at its maximum value, increasing the other parameter will generally increase their impact further. Because they are relatively similar in terms of their requirements for causing strong reservation conflicts, however, their combined effects are not simply the sum of their individual effects; every traffic situation in which two vehicles are planning conflicting maneuvers can potentially lead to a strong reservation conflict due to one of the parameters, but never both at the same time.

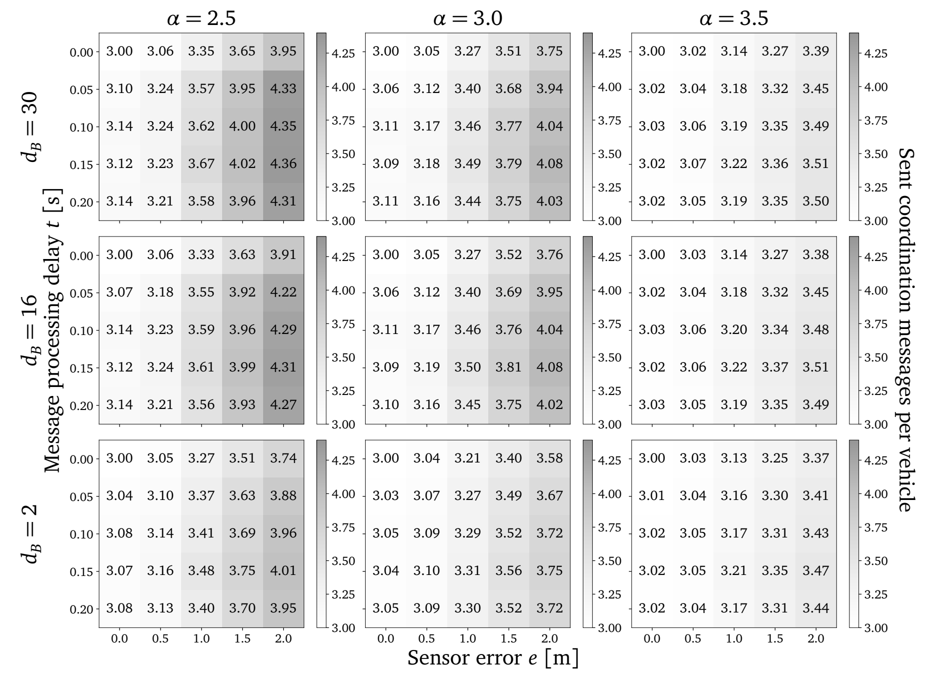


Figure 5.27 – Average number of sent coordination messages per vehicle in the combined sensor error and processing delay simulation.

Chapter 6

Conclusion

In this thesis, I implemented the vehicle controller for provably safe crossing maneuvers at urban intersections, which was introduced by Schwammberger [1], in the vehicular network simulation framework Veins. The main goal was the evaluation of the controller's performance in terms of safety under more realistic conditions than the highly abstract traffic model in which the controller is defined. To reach this goal, I implemented various mechanisms for introducing imperfections that vehicles are commonly exposed to in reality and conducted extensive simulation studies for a variety of intersection layouts, investigating numerous safety performance metrics. I documented the implementation process to report on design problems and discussed the simulation results in great detail to provide appropriate explanations of all findings.

The main result is that the crossing controller works as intended and satisfies the promised safety property in the most abstract simulation setting, which is already much more realistic than the abstract traffic model. Although the implementation process revealed a small number of design problems related to the discrepancy between the high level of abstraction and the concrete implementation details, I was able to leverage the abstraction level to solve these problems without significantly altering the controller's semantics. The most problematic aspect of the controller definition in this regard was the lack of an execution model: As the abstract traffic model and the formal logic UMLSL only allow for spatial reasoning, it is the responsibility of the developer to design the temporal component of the control system. While the abstraction level and the relatively simple controller definition facilitate a modular architecture that supports the usage of already existing components, they can also be the cause of ambiguities and unexpected problems such as blocked intersections.

The simulations showed that effects like limited communication possibilities, random errors in the perception of other vehicles' positions, and communication delays can compromise the controller's safety property and lead to collisions. A variety of additional factors such as the intersection layout, the amount of traffic, and the distance to the intersection at which the vehicles start their coordination procedure can greatly affect the severity of these effects, both positively and negatively. However, I found suitable explanations for most of these relations by using multiple performance metrics and a simple communication model that is easy to reason about. More importantly, all of the safety problems introduced by the simulation parameters can be attributed to this communication protocol or other implementation design decisions that are not enforced by the controller definition itself. This means that a more sophisticated communication protocol or execution model could increase the robustness and reliability of the controller significantly.

All of these findings lead to the conclusion that the crossing controller, as defined in the abstract traffic model, cannot fully transfer its safety property into a realistic environment in which the simplifying, idealizing assumptions about the world do not hold, but it provides all the necessary potential to be the basis for a concrete implementation that *does* possess the same level of guaranteed safety. The abstract traffic model and UMLSL are meant to be used as tools for defining arbitrary vehicle controllers and formally proving that they satisfy the desired properties. Their high level of abstraction also provides the opportunity to refine a control system by adding components like communication protocols and proving more properties like liveness or fairness until the controller meets its requirements and can be implemented more easily.

As an example for this, the version of the crossing controller that was extended by a communication protocol to handle imperfect information [5] could be evaluated in future work similar to this thesis. Alternatively, it could be interesting to extend the crossing controller implementation I presented here by an already existing communication protocol or vehicular networking infrastructure to investigate to what extent this would improve the controller's performance. Additionally, there are various ways to further increase the degree of realism in the simulation scenarios. For example, multiple vehicle types with different acceleration values and individual communication ranges, sensor errors, and processing delays could be used to model the diversity of real traffic, which was not considered in this thesis. Similarly, the scenarios themselves could be extended to include multiple intersections with connection roads, or they could even be modeled after real traffic networks using geographical data in order to investigate large-scale city scenarios.

List of Abbreviations

ACTA	Automotive-Controlling Timed Automaton
CACC	Cooperative Adaptive Cruise Control
IVC	inter-vehicular communication
MLSL	Multi-lane Spatial Logic
UMLSL	Urban Multi-lane Spatial Logic
VANET	Vehicular Ad Hoc Network

List of Figures

2.1 Abstract intersection representation 8
2.2 Crossing controller automaton 15
2.3 Vehicle chape computation in SUMO 18

3.1 Comparison of maneuver representations 24
3.2 Example intersection from SUMO 28
3.3 Computing internal lane shapes 29
3.4 Foe matrix entries for overlapping internal lanes 31
3.5 Computation of reservation clearance positions 32
3.7 Livelock example with two vehicles 39

4.1 Scenario generation with SUMO 47
4.2 Placement of buildings in the simulation scenario 53
4.3 Relative crash velocity calculation 58

5.1 Baseline simulation overview 66
5.2 Strong reservation conflicts in unit disk model (medium traffic demand) 68
5.3 Communication range visualization 68
5.4 Strong reservation conflicts in unit disk model (low and high traffic demand) 69
5.5 Strong reservation conflicts in simple path loss model (medium traffic demand) 70
5.6 Strong reservation conflicts and collisions in obstacle shadowing model (medium traffic demand) 72
5.7 Increasing the building distance – visualization 73
5.8 Strong reservation conflict time stamps in obstacle shadowing model 74
5.9 Strong reservation conflicts in obstacle shadowing model (low and high traffic demand) 75
5.10 Trip times in obstacle shadowing model (medium traffic demand) . . 76

5.11 Strong reservation conflicts for message loss parameter (medium traffic demand) . . . 77
5.12 Collisions for message loss parameter (medium traffic demand) . . . 78
5.13 Trip times for message loss parameter (medium traffic demand) . . . 79
5.14 Squared relative crash velocity for message loss parameter (medium traffic demand) . . . 80
5.15 Strong reservation conflicts with sensor error parameter (all traffic demands) . . . 82
5.16 Sent messages with sensor error parameter (medium traffic demand) 83
5.17 Trip times for sensor error parameter (medium traffic demand) 84
5.18 Strong reservation conflicts and collisions with processing delay (all traffic demands) . . . 85
5.19 Processing delay sequence diagram . . . 86
5.20 Relation between controller trigger interval and processing delay . . 87
5.21 Sent coordination messages with processing delay (medium traffic demand) . . . 87
5.22 Strong reservation conflicts with simple path loss and obstacle shadowing model (all traffic demands) . . . 89
5.23 Strong reservation conflicts in combined sensor error simulation . . . 91
5.24 Sent messages in combined sensor error simulation . . . 92
5.25 Collisions in combined processing delay simulation . . . 93
5.26 Strong reservation conflicts in combined simulation of all parameters 94
5.27 Sent coordination messages in combined simulation of all parameters 95

List of Tables

4.1 Turning maneuver probability distribution 50

5.1 Simulation scenario overview 64

Bibliography

[1] M. Schwammberger, “An abstract model for proving safety of autonomous urban traffic,” *Elsevier Theoretical Computer Science*, vol. 744, pp. 143–169, Oct. 2018. DOI: 10.1016/j.tcs.2018.05.028.

[2] D. Jia, K. Lu, J. Wang, X. Zhang, and X. Shen, “A Survey on Platoon-Based Vehicular Cyber-Physical Systems,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 263–284, 2016. DOI: 10.1109/COMST.2015.2410831.

[3] M. Hilscher, S. Linker, E.-R. Olderog, and A. P. Ravn, “An Abstract Model for Proving Safety of Multi-lane Traffic Manoeuvres,” in *13th International Conference on Formal Engineering Methods (ICFEM 2011)*, Durham, United Kingdom: Springer, Oct. 2011, pp. 404–419. DOI: 10.1007/978-3-642-24559-6_28.

[4] M. Hilscher and M. Schwammberger, “An Abstract Model for Proving Safety of Autonomous Urban Traffic,” in *13th International Colloquium on Theoretical Aspects of Computing (ICTAC 2016)*, Taipei, Taiwan: Springer, Oct. 2016, pp. 274–292. DOI: 10.1007/978-3-319-46750-4_16.

[5] M. Schwammberger, “Imperfect Knowledge in Autonomous Urban Traffic Manoeuvres,” in *1st Workshop on Formal Verification of Autonomous Vehicles (FVAV@iFM 2017)*, vol. 257, Turin, Italy: Open Publishing Association, Sep. 2017, pp. 59–74. DOI: 10.4204/eptcs.257.7.

[6] C. Sommer, R. German, and F. Dressler, “Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis,” *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, Jan. 2011. DOI: 10.1109/TMC.2010.133.

[7] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil, “Vehicular Networking: A Survey and Tutorial on Requirements, Architectures, Challenges, Standards and Solutions,” *IEEE Communications Surveys & Tutorials*, vol. 13, no. 4, pp. 584–616, Nov. 2011. DOI: 10.1109/SURV.2011.061411.00019.

[8] L. Chen and C. Englund, "Cooperative Intersection Management: A Survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 2, pp. 570–586, Feb. 2016. DOI: `10.1109/TITS.2015.2471812`.

[9] E. Namazi, J. Li, and C. Lu, "Intelligent Intersection Management Systems Considering Autonomous Vehicles: A Systematic Literature Review," *IEEE Access*, vol. 7, pp. 91 946–91 965, Jul. 2019. DOI: `10.1109/ACCESS.2019.2927412`.

[10] J. Dahl, G. R. de Campos, C. Olsson, and J. Fredriksson, "Collision Avoidance: A Literature Review on Threat-Assessment Techniques," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 1, pp. 101–113, Mar. 2019. DOI: `10.1109/TIV.2018.2886682`.

[11] K. Dresner and P. Stone, "Multiagent Traffic Management: A Reservation-Based Intersection Control Mechanism," in *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, vol. 2, New York City, NY: IEEE Computer Society, Jul. 2004, pp. 530–537. DOI: `10.1109/AAMAS.2004.10121`.

[12] ——, "A Multiagent Approach to Autonomous Intersection Management," *Journal of Artificial Intelligence Research (JAIR)*, vol. 31, no. 1, pp. 591–656, Jan. 2008. DOI: `10.1613/jair.2502`.

[13] H. Kowshik, D. Caveney, and P. R. Kumar, "Provable Systemwide Safety in Intelligent Intersections," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 3, pp. 804–818, Mar. 2011. DOI: `10.1109/TVT.2011.2107584`.

[14] M. R. Hafner, D. Cunningham, L. Caminiti, and D. Del Vecchio, "Automated Vehicle-to-Vehicle Collision Avoidance at Intersections," in *18th World Congress on Intelligent Transport Systems (ITS 2011)*, Orlando, FL, Oct. 2011.

[15] ——, "Cooperative Collision Avoidance at Intersections: Algorithms and Experiments," *IEEE Transactions on Intelligent Transportation Systems (TITS)*, vol. 14, no. 3, pp. 1162–1175, Sep. 2013. DOI: `10.1109/TITS.2013.2252901`.

[16] M. Bashiri and C. H. Fleming, "A platoon-based intersection management system for autonomous vehicles," in *IEEE Intelligent Vehicles Symposium (IV) 2017*, Redondo Beach, CA: Institute of Electrical and Electronics Engineers (IEEE), Jun. 2017, pp. 667–672. DOI: `10.1109/IVS.2017.7995794`.

[17] M. Bashiri, H. Jafarzadeh, and C. H. Fleming, "PAIM: Platoon-based Autonomous Intersection Management," in *21st International Conference on Intelligent Transportation Systems (ITSC 2018)*, Maui, HI: Institute of Electrical and Electronics Engineers (IEEE), Nov. 2018, pp. 374–380. DOI: `10.1109/ITSC.2018.8569782`.

[18] M. Hilscher, S. Linker, and E.-R. Olderog, “Proving Safety of Traffic Manoeuvres on Country Roads,” in *Theories of Programming and Formal Methods: Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday*, Z. Liu, J. Woodcock, and H. Zhu, Eds. Shanghai, China: Springer, Sep. 2013, pp. 196–212. DOI: `10.1007/978-3-642-39698-4_12`.

[19] R. Alur and D. L. Dill, “A theory of timed automata,” *Elsevier Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, Apr. 1994. DOI: `10.1016/0304-3975(94)90010-8`.

[20] A. Varga and R. Hornig, “An Overview of the OMNeT++ Simulation Environment,” in *1st ACM/ICST International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008)*, Marseille, France: Association for Computing Machinery (ACM), Mar. 2008.

[21] P. Alvarez Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, “Microscopic Traffic Simulation using SUMO,” in *21st IEEE International Conference on Intelligent Transportation Systems (ITSC 2018)*, Maui, HI: Institute of Electrical and Electronics Engineers (IEEE), Nov. 2018, pp. 2575–2582. DOI: `10.1109/ITSC.2018.8569938`.

[22] M. Fränzle, M. R. Hansen, and H. Ody, “No Need Knowing Numerous Neighbours,” in *Correct System Design*, Proceedings of the Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, R. Meyer, A. Platzer, and H. Wehrheim, Eds., vol. 9360 of LNCS, Oldenburg, Germany: Springer, Sep. 2015, pp. 152–171. DOI: `10.1007/978-3-319-23506-6_11`.

[23] S. Krauß, “Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics,” PhD Thesis, Mathematical Institute, Köln, Germany, Apr. 1998.

[24] C. Sommer, D. Eckhoff, R. German, and F. Dressler, “A Computationally Inexpensive Empirical Model of IEEE 802.11p Radio Shadowing in Urban Environments,” in *8th IEEE/IFIP Conference on Wireless On demand Network Systems and Services (WONS 2011)*, Bardonecchia, Italy: Institute of Electrical and Electronics Engineers (IEEE), Jan. 2011, pp. 84–90. DOI: `10.1109/WONS.2011.5720204`.

[25] T. L. Willke, P. Tientrakool, and N. F. Maxemchuk, “A Survey of Inter-Vehicle Communication Protocols and Their Applications,” *IEEE Communications Surveys & Tutorials*, vol. 11, no. 2, pp. 3–20, Jun. 2009. DOI: `10.1109/SURV.2009.090202`.

[26] J. Miranda, R. Abrishambaf, T. Gomes, J. Cabral, A. Tavares, and J. Monteiro, "Path Loss Exponent Analysis in Wireless Sensor Networks: Experimental Evaluation," in *11th IEEE International Conference on Industrial Informatics (INDIN)*, Bochum, Germany: Institute of Electrical and Electronics Engineers (IEEE), Jul. 2013, pp. 54–58. DOI: 10.1109/INDIN.2013.6622857.