Group Communication for Cooperative Automated Vehicles

Martin Cornelis Frederik Slot

A thesis submitted to the University of Dublin, Trinity College

in fulfilment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

2014

Declaration

I, the undersigned, declare that this work has not previously been submitted to this or any other University, and that unless otherwise state, it is entirely my own work. I agree that Trinity College Library may lend or copy this thesis upon request.

[illegible]

Martin Cornelis Frederik Slot

Dated:

Thesis 10803

Acknowledgements

In early 2008, I was often dreaming of a new idea. What if cars could form a distributed system over a wireless network for controlling the cars themselves? Vehicles could move around smoothly like flocks of birds, never hitting each other, coordinating every move over a network they themselves create. Cooperative self-driving cars could drive in ways that human drivers could never achieve. The implications on the safety, efficiency, and comfort of driving could be enormous. What a fun, challenging, and relevant problem that would be to work on. I feel extremely fortunate that my supervisor, Vinny Cahill, saw this problem many years earlier, and privileged that I could work on it under his supervision at such a wonderful place as Trinity College Dublin. My sincerest gratitude goes out to him for the opportunity and his insightful guidance. I recognize that many of the foundations of my own research were laid by other researchers in the Distributed Systems Group before I even started. I would also like to thank Mikael Asplund and Mélanie Bouroche, for their very helpful co-supervision, my M.Sc. supervisor, Guillaume Pierre, for sending me off to a flying start in the world of research, and David Richardson and Amazon.com, for giving me a wonderful job to do next to my research, to keep me thoroughly distracted, as well as inspired. I would also like to thank all my friends who have been enormously supportive throughout the years. Most of all, I would like to thank my parents, Chiel and Catja, for their unremitting support, my sister Cynthia, for keeping me going when things were tough, as well as Manfred, Milan, and Collin, for giving me a great reason to come home.

Publications Related to this Ph.D.

As of April 2013, parts of this thesis have appeared in the following publications:

- Marco Slot, Mélanie Bouroche, Vinny Cahill.
  Membership service specifications for safety-critical geocast in vehicular networks.
  In IEEE, IET International Symposium on Communication Systems Networks and Digital Signal Processing, 2010, pages 422-426

- Marco Slot, Vinny Cahill.
  A reliable membership service for vehicular safety applications.
  In IEEE Intelligent Vehicles Symposium, 2011, pages 1163-1169

- Marco Slot, Vinny Cahill.
  End-to-end acknowledgement of geocast in vehicular networks.
  In IEEE Vehicular Networking Conference, 2011, pages 131-138

- Marco Slot, Mélanie Bouroche, Vinny Cahill.
  Design of a safety-critical geocast API for vehicular coordination.
  In International Journal of Vehicle Information and Communication Systems, 2011, Vol.2, No.3/4, pages 193-212

Abstract

The automotive industry has increasingly adopted computer technology to enhance the safety and efficiency of vehicles. Using sensors and electronic control systems, vehicles can become partially or fully automated, taking only high-level instructions from the driver and making the necessary driving decisions themselves. Through communication over a wireless (vehicular) network, automated vehicles can cooperate by sharing sensor information and coordinating their driving decisions with each other in order to improve the safety and efficiency of driving. However, the problem of having automated vehicles cooperate in a safe, distributed manner is mostly unsolved. One of the biggest challenges is to coordinate safety-critical driving decisions over a wireless network in the presence of omission failures, messages being lost in transmission. Many existing cooperative driving solutions are unsuitable for fully automated driving, since they do not tolerate omission failures and rely on a human driver to intervene. Other solutions rely on the network to have a bounded number of omission failures, a dangerous assumption in an environment as dynamic as that of vehicular networks.

Omission failures are a common problem in networked systems and are often dealt with through acknowledgements, receipt confirmations that allow senders to confirm successful delivery of a message to a communication group. An application can proceed if communication is successfully acknowledged, and adapt otherwise, taking into account possible failure scenarios that could have occurred. However, in cooperative automated driving applications, communication groups change with the movements of vehicles and group membership cannot be known reliably in the presence of omission failures. Without a reliable view of the communication group, acknowledgements are insufficient to confirm successful delivery. Fortunately, as this thesis will show, a combination of communication and sensors can be used to create a membership view that enables reliable success confirmation for communication to geographically-defined groups of vehicles, even in the presence of unbounded omission failures and inaccurate sensors.

This thesis introduces the Vertigo communication model, which defines a spatio-temporal model of group membership, and the interface to a group communication system (GCS), a building block for distributed applications which offers many-to-many communication services. In the Vertigo model, the GCS offers a geocast operation for sending a message to, and gathering responses from, all vehicles that are present in a given target area at a given target time, and reliably confirms successful delivery whenever possible. Applications can proceed if success is confirmed, and adapt to possible failure scenarios in the absence of confirmation. The feasibility of implementing the model is shown through an implementation using sensing and communication capabilities that are common in automated vehicles. The applicability of the Vertigo model is demonstrated through the implementation of a safe, distributed coordination protocol that uses the model. Simulations show that even with limited sensing capabilities, the Vertigo implementation can achieve a rate of success that is sufficiently high for the coordination protocol to significantly outperform a traditional traffic management approach. These results show the Vertigo model as a viable solution for implementing a safe, efficient, cooperative automated driving system.

Contents

Acknowledgements iii

Abstract iv

List of Tables xii

List of Figures xiii

Chapter 1 Introduction 1

1.1 Motivation 1

1.2 Group Communication for Cooperative Automated Cars 3

1.3 Reliable, Spatio-temporal Membership 3

1.4 Geocast with Reliable Success Confirmation 5

1.5 Crossing an Intersection using Vertigo 6

1.6 Evaluation 6

1.7 Philosophy 8

1.8 Roadmap 9

Chapter 2 Related Work 10

2.1 Cooperative Automated Driving 10

2.1.1 Automated Highway Systems 11

2.1.2 Intersection Collision Avoidance 13

2.2 Vehicular Networks 15
2.3 Geographic Communication in Vehicular Networks 16
2.3.1 Reliable Communication in Vehicular Networks 19
2.4 Group Communication Systems 20
2.4.1 Real-time and Mobile Group Communication 21
2.4.2 Reliable Group Communication 22
2.5 Discussion 23

Chapter 3 System Model **25**
3.1 Design principles 25
3.2 Cooperative Automated Driving Requirements 27
3.2.1 Safety constraints 27
3.2.2 Speed and locality 28
3.2.3 Actions 28
3.2.4 Automated driving 29
3.2.5 Cooperative automated driving 29
3.3 Membership model 30
3.3.1 Membership tuples 30
3.3.2 Generating Tuples 31
3.3.3 Decaying tuples 32
3.3.4 Merging tuples 32
3.3.5 Discussion 33
3.4 Group Communication Model 34
3.4.1 Geocast 34
3.4.2 Timeliness 36
3.4.3 Feedback with membership 38
3.4.4 Sender API 38
3.4.5 Receiver API 41

3.4.6 Discussion 43

Chapter 4 Design and Implementation **44**

4.1 Capabilities 46

4.2 Road Map Area Definition 46

4.3 Beaconing service 49

4.4 Membership Service 49

4.4.1 Sensing a Local Membership Area 49

4.4.2 Beaconing Membership Tuples 57

4.4.3 Merging 1D Road Map Areas 57

4.4.4 Decaying 1D Road Map Areas 58

4.4.5 Collapsing Membership Tuples 63

4.5 Group Communication Service 64

4.5.1 Starting a geocast query 64

4.5.2 Receiving and responding to a geocast query 64

4.5.3 Feedback on a geocast query 66

4.5.4 Forwarding 67

4.6 Discussion 70

Chapter 5 Application **71**

5.1 Tracks 71

5.2 Conflict Areas 72

5.3 Empty Trajectory 72

5.4 Moving Forward 73

5.5 Coordination Protocol 74

5.5.1 Initiating the Coordination Protocol 74

5.5.2 Conflict Trajectory 75

5.5.3 Allocation State 76

5.5.4 Sending an Allocation Request 76

5.5.5 Receiving an Allocation Request 78
5.5.6 Obtaining an Allocation 80
5.5.7 Spatial Commit 81
5.5.8 Resolving Allocation Dependencies 81
5.6 Formal Analysis 83
5.7 Discussion 87

Chapter 6 Evaluation **88**
6.1 Evaluation strategy 88
6.2 Simulator 89
6.2.1 Driving 92
6.2.2 Sensors 93
6.2.3 Wireless ad-hoc network 93
6.2.4 Scenario 94
6.2.5 Controller 95
6.3 Experiments 96
6.3.1 LIDAR range 98
6.3.2 Position accuracy 102
6.3.3 Offered rate of entry 105
6.3.4 WLAN Transmission power 108
6.4 Discussion 110

Chapter 7 Conclusions **112**
7.1 Contributions 112
7.2 Future Work 114

Appendix A API Notation **117**

Appendix B Lower-layer APIs **119**
B.0.1 Ad-hoc networking API 119

B.0.2 Sensor APIs 120

Appendix C Road map specifications **124**

C.1 Road segment graph 124

C.2 Position specification 128

C.3 Area specification 129

C.3.1 Boundary-based representation 129

C.3.2 Range-based representation 130

C.3.3 Boundaries to ranges translation 131

C.3.4 Ranges to boundaries translation 135

Bibliography **136**

List of Tables

3.1 Design principles 26
3.2 Sender API specifications 39
3.3 Receiver API specifications 41

5.1 Scenarios for concurrent allocation requests 85

6.1 Default values for simulation parameters 97

A.1 API notation 117

B.1 Communication API specifications 120
B.2 Position sensor specifications 121
B.3 Orientation sensor specifications 122
B.4 Ranging sensor specifications 122

C.1 Road map specifications 124
C.2 Position specifications 128
C.3 Area representation for messages 130
C.4 Internal area representation 131

List of Figures

4.1 Architecture of the Vertigo implementation 45
4.2 Road map representation used by the Vertigo implementation consisting of road segments and connectors 47
4.3 Area representations used in the Vertigo implementations 48
4.4 LIDAR Sensors on the vehicle 50
4.5 Converting LIDAR beams to a ring of points 51
4.6 Conversion of LIDAR data (green) to polygon (yellow) 52
4.7 Intersection of a measured polygon with the road map 54
4.8 Conversion of 2D road map area to 1D 55
4.9 Conversion of polygon to one-dimensional ranges 56
4.10 An example of decay of a one-dimensional road map area 59
4.11 A successful query, the membership area (green) covers the target area (blue) 67
4.12 Unsuccessful query due to omission failure, the membership area (red) does not cover the target area (blue) 68

5.1 Conflict areas at the intersection of tracks 72
5.2 Empty trajectory (green) provides forward distance 73
5.3 A conflict trajectory through 2 conflict areas with a commit area of length l_{commit} and stopping distance $l_i + d_{min} + l_{commit}$ 76

5.4 The allocation state machine exemplified with a conflict trajectory 77

5.5 Delivery area (blue) expanded from conflict areas 79

5.6 Scenarios for concurrent allocation requests by vehicles i and j 84

6.1 A simulation in roundasim with cars shown in red 90

6.2 Tracks and buildings on the Merrion Square intersection shown in gray . . 94

6.3 The average success rate of Vertigo queries for different LIDAR ranges . . 99

6.4 The average network overhead (over all vehicles) for different LIDAR ranges 99

6.5 The average rate of traffic over the intersection for different LIDAR ranges 100

6.6 Quartiles of the travel time over the intersection for different LIDAR ranges 100

6.7 An example of position inaccuracy being too high to continue 102

6.8 The average success rate of Vertigo queries for different position inaccuracies 103

6.9 The average network overhead (over all vehicles) for different position inaccuracies 103

6.10 The average rate of traffic for different position inaccuracies 104

6.11 Quartiles of the travel time for different position inaccuracies 104

6.12 The average success rate of Vertigo queries for different entry rates 106

6.13 The average network overhead (over all vehicles) for different entry rates . 106

6.14 The average rate of traffic over the intersection for different entry rates . . 107

6.15 Quartiles of the travel time over the intersection for different entry rates . 107

6.16 The average success rate of queries for different transmission powers . . . 109

6.17 Quartiles of the travel time for different transmission powers 109

Chapter 1

Introduction

This thesis presents Vertigo, a real-time vehicular group communication model for co-operative automated driving applications, and its implementation. Vertigo provides automated vehicles with the ability to safely coordinate their decisions with other vehicles through a communication model that offers geocast communication with reliable success confirmation. This chapter describes the motivation for the work and the contributions of this thesis, as well as a road map for the remainder of the thesis.

1.1 Motivation

Automated driving is a development that could have significant impact on transportation. Traffic jams and accidents could be dramatically reduced, people of all ages and abilities will be able to drive, driverless taxis could become cheap and abundant, and private vehicles could pick up and drop off their owners before parking in compact car parks. Developments in automated driving are moving quickly, boosted by the DARPA grand challenge [Urmson 08], Google's autonomous car [Levinson 11], academia [Luettel 12], and car manufacturers [Jaynes 13]. Research vehicles can drive automatically through cities and on highways, so far with an excellent safety record.

At present, most automated vehicles have been strictly autonomous. They make driving decisions based purely on information from their own sensors, which may be noisy, limited in range, and difficult to process [Urmson 08]. Vehicular networks could enable automated vehicles to cooperate by announcing their presence over the network and sharing sensor information, as well as coordinating driving decisions with each other [Tsugawa 02]. Cooperation can make automated driving safer and more efficient, since cooperative vehicles can plan their actions prior to observing each other on sensors, and do not have to rely on noisy sensor data to detect other vehicles or their intentions.

Cooperative automated vehicles form a distributed system in which vehicles need to achieve common goals, while having only partial knowledge of each other's state. Distributed systems are well-studied in environments such as wired networks [Lynch 96], the Internet [Steen 12], and sensor networks [Hadim 06], but vehicles operate in a very different environment and have a very different set of requirements.

Controlling a vehicle requires decision algorithms that are both time-critical, meaning that the decision is made before a deadline, and safety-critical, meaning that the decision may never lead to a violation of safety constraints. However, the communication channel between vehicles is unreliable. Unbounded omission failures can occur with positive probability. This problem is left unaddressed by existing communication solutions for automated vehicles that either assume a communication channel with a bounded number of omission failures [Nett 03], provide only probabilistic guarantees [Ros 09], or remove vehicles that fail to communicate from the system [Maxemchuk 07].

A pure communication solution that can be relied upon by safety-critical applications is impossible under unbounded omission failures as messages may not arrive, and acknowledgements are ineffective, as the group of intended receivers may not be known. Fortunately, as this thesis will show, a combination of sensors and communication is sufficient for implementing safe decision algorithms, even in the presence of unbounded omission failure and inaccurate sensors, through the use of the Vertigo communication model.

1.2 Group Communication for Cooperative Automated Cars

To simplify the development of distributed systems, an often applied technique is a group communication system (GCS) [Chockler 01]. A GCS is middleware that provides many-to-many communication services with certain guarantees to applications in terms of ordering and message delivery. A central component of a GCS is a membership service that provides a view of membership of a communication group. The GCS offers guarantees for communication operations within a particular membership view.

This thesis introduces a new model for group communication and membership that specifically addresses communication between cooperative automated vehicles. In particular, it provides a method of performing geocast, broadcast to all nodes within a geographic area, with safety-critical, time-critical success confirmation. The model is designed to bridge the gap between application requirements (safe progress) on the one hand, and the abilities of automated vehicles (sensors and communication) and inherent limitations imposed by vehicular networks and distributed systems (unreliability, delay) on the other. The combined group communication and membership model are referred to as the Vertigo model, which stands for Vehicular, Real-Time Group Communication model.

1.3 Reliable, Spatio-temporal Membership

The common way of representing a membership view for a GCS is as a set of node identifiers (members) with a version number [Chockler 01]. The Vertigo model uses geographically-defined communication groups. A membership view is represented as a tuple of a set of identifiers M, an area A, and a point in time t, denoted $\mathcal{M}(M, A, t)$. The meaning of the tuple, and simultaneously the guarantee that an implementation of the Vertigo model must offer with regards to tuples, is that the identifier of any vehicle that is present in the membership area A at the membership time t is in the set of identifiers (members) M. This guarantee is central to the Vertigo model. It follows from the guarantee that if a vehicle sends a message and receives a response from all members in M, then it knows that all vehicles in area A at time t have been reached. The guarantee can be provided with safety-critical reliability in spite of communication delay, inaccurate and incomplete sensor data, and unreliable communication.

Membership tuples can be formed incrementally. This process is performed by *decaying* and *merging* tuples. Decay shifts the membership time forward by δ seconds. To preserve the guarantee that any vehicle in the membership area at the membership time is in the set of members, the membership area is reduced in size to exclude any point that vehicles could have reached if they had driven into the area at maximum speed for δ seconds. Merging combines two tuples from the same time, taking the union of the membership sets and areas. For any two tuples, the older tuple can be decayed to the membership time of the younger tuple and subsequently merged, which allows incremental construction of tuples from an arbitrary set of smaller tuples.

The elementary tuples from which larger tuples can be constructed can be formed reliably using sensors. The implementation of the membership service uses LIDAR to establish an empty area A around a vehicle i at time t that is reduced in size to account for worst-case sensor inaccuracy bound. From the area that is empty except for the vehicle itself, a tuple $\mathcal{M}(\{i\}, A, t)$ is constructed, which is then communicated to other vehicles. To form a membership view, vehicles merge tuples generated using their own sensors with those received from other vehicles until a tuple is formed that covers a given set of identifiers or a given area.

Membership tuples provide a unique ability for applications to reason about the state of all vehicles in an area A_{target} at time t_{target}. A membership tuple $\mathcal{M}(M, A, t_{target})$, whose membership area A contains A_{target}, is effectively a proof of the completeness of M as a superset of the identifiers of vehicles in area A_{target} at time t_{target}. Anything that holds true for all vehicles whose identifier is in M, must also hold true for all vehicles in A_{target} at time t_{target}. This property of membership tuples is used in the Vertigo communication model to confirm that a message is delivered to all vehicles in a given target area at a given target time.

1.4 Geocast with Reliable Success Confirmation

The group communication model of Vertigo is defined as an API for geocast with feedback. The geocast operation takes a message, a target area A_{target}, and a set of time constraints as parameters. The time constraints define when the message should be delivered to applications, when feedback on success of the geocast should be provided to the application, and the target time defines the set of intended receivers as all vehicles present in the target area at the target time.

The geocast operation disseminates the message to all vehicles that might be in a target area A_{target} at a target time t_{target}. When the message is received by a vehicle, it is delivered to an application, which can then decide to respond. Responses are sent back to the sender of the geocast. Feedback on the success of the geocast is provided to the sender at the configured time and contains the set of responders R and a membership tuple $\mathcal{M}(M, A, t_{target})$. Success is reliably confirmed if $R \supseteq M \wedge A \supseteq A_{target}$, meaning the set of confirmed responders is a superset of the vehicle in area A_{target} at t_{target}. Applications using the geocast can adapt to whether or not communication was successful. In safety-critical applications, absence of success confirmation should be treated as a failure.

The implementation of the communication model uses an ad-hoc network interface based on 802.11p [IEEE 10]. The geocast message is broadcast over the ad-hoc network and received by vehicles that are present in the delivery area that is attached to the message. The delivery area is derived from the target area by expanding it to anticipate the effects of decay. The delivery area needs to be big enough such that all vehicles that will be in the target area at the target time are already in the delivery area at the start of the geocast. Responses to the geocast are sent back over the ad-hoc network using unicast, and membership information is disseminated using beaconing.

1.5 Crossing an Intersection using Vertigo

To demonstrate the applicability of the Vertigo model to the problem of cooperative automated driving, a coordination protocol that uses Vertigo is implemented. The protocol uses a road network model that is based on tracks, interconnected line strings that vehicles follow. Where two tracks cross, a conflict area exists, and mutual exclusion needs to be guaranteed for conflict areas.

Once within a certain distance of a set of conflict areas, a vehicle tries to obtain an allocation that allows entry into the conflict areas during a specific time frame. To do so, a vehicle sends an allocation request using Vertigo to an area containing all vehicles that might be trying to obtain an allocation for the conflict area at the same time or already have one. Receivers of the request send back a response, which indicates whether they reject, accept, or tentatively accept the allocation. Tentative responses create a distributed dependency graph between allocations, determining in which order vehicles can pass a conflict area. If all vehicles accept a request and successful communication can be confirmed using the Vertigo model, the sender can proceed into the conflict areas, once it has confirmed that all of the vehicles on which its allocation depends have moved past the conflict area or are no longer in the area at all. Membership can be used to reliably detect the absence of a vehicle in case of omission failures. The correctness of the protocol is verified to demonstrate that the Vertigo model can indeed support *safe*, distributed coordination protocols.

1.6 Evaluation

The implementations of the coordination protocol and Vertigo are evaluated using a novel simulator for cooperative automated driving scenarios. The simulator provides simulation of GPS and LIDAR, and uses SWANS [Barr 05] to simulate a wireless ad-hoc network between the vehicles, and the Intelligent Driver Model [Treiber 13] to control their velocities.

Neither the Vertigo implementation, nor the coordination protocol can be studied in isolation. Progress in the coordination protocol can be quantified as the rate at which vehicles move through the scenario, which directly depends on the rate at which Vertigo confirms successful delivery. At the same time, the success rate of Vertigo depends on the positions and actions of vehicles that result from the coordination protocol and the scenario. Additionally, using a safe coordination protocol is essential to the evaluation of Vertigo, since LIDAR could not be correctly simulated if vehicles were ever allowed to overlap (crash). The protocol also generates a realistic usage pattern of Vertigo queries. The Vertigo implementation and coordination protocol are therefore evaluated in tandem by using them to manage a four-way intersection.

Through a series of parameter studies, bounds on position inaccuracy, LIDAR range, and traffic density, are found representing the worst-case conditions under which Vertigo can achieve a success rate that still allows the coordination protocol to maintain a high rate of traffic. Position inaccuracy of up to 3m is shown to be tolerable, which is much more permissive than what is normally required for automated vehicles (less than 0.1m) [de Nooij 10] and state-of-the-art positioning for automated vehicles achieve much greater accuracy [Elkaim 08]. Thanks to the cooperative nature of the membership service, a maximum LIDAR range as short as 15m is sufficient to allow progress over the intersection at a reduced rate of traffic, while a maximum range of 25m is sufficient for achieving maximal traffic rates. Both ranges fall well within the specifications of LIDARs for automated vehicles [Buehler 09].

Vertigo scales to high traffic densities and is only limited by the rate at which vehicles can cross the intersection. The cooperative automated vehicles using Vertigo achieve a maximum throughput on the intersection that lies roughly 25% higher than that of ordinary traffic lights. These results show the feasibility of building an implementation of the Vertigo model that can be used to solve a real cooperative automated driving problem in a way that guarantees safety, while also improving efficiency.

1.7 Philosophy

Over the course of the research that leads to a thesis, a question worth asking is: 'What is Computer Science?'. The way in which science is performed is determined by the philosophy of those performing it, and thus is core to the motivation of the way in which this and all other research is performed. The insights in this section are neither new nor extraordinary, but they emphasize how the contributions of this thesis should be viewed.

What puts Computer Science apart from most other fields, is that its subjects of study are man-made. There was no universal process that led to all computer systems. Instead, every system is individually crafted. This might prevent us from ever finding simple models that are common to all systems and being able to understand their behaviour in a fundamental way. However, the fact that computer systems are man-made provides us with the possibility of reversing the process: to develop systems around simple models that help us reason about them.

When entering into a new class of problems, such as cooperation between automated vehicles, Computer Science should search for models that provide an appropriate mapping between the reality of the environment and the functions required to address problems from the problem space, such as safety under unbounded communication failure. Whether a model truly has merit, either by being used in real-world systems or by inspiring other models, will only be determined by time. However, a basic test of legitimacy can be performed by demonstrating that it is *feasible* to implement and *applicable* to the problem space.

Feasibility can be demonstrated through an implementation of the model that is able to fulfil its requirements. In the particular case of cooperative automated driving, feasibility also concerns the ability to function in a physical environment, as the computational environment (e.g. the network topology) is directly influenced by the physical environment (e.g. cars moving). Applicability can be demonstrated through an implementation of one or more algorithms that use the model to solve a real problem from the problem space that the model addresses.

This thesis introduces a new group communication model for addressing cooperative automated driving problems, and validates it by demonstrating its feasibility and applicability. Feasibility is shown by providing an implementation of the model that achieves a high rate of success using basic network and sensor interfaces. Applicability is shown by providing an implementation of a cooperative automated driving algorithm that uses the model to cooperate in a way that guarantees safety using the properties of the model, while outperforming a traditional traffic management approach.

1.8 Roadmap

The remainder of this thesis has the following structure:

- Chapter 2 discusses the state-of-the-art in cooperative automated driving and communication systems in support of it.
- Chapter 3 derives and defines the Vertigo model comprising a membership and group communication model.
- Chapter 4 presents a set of protocols and algorithms to implement the Vertigo model.
- Chapter 5 introduces a coordination protocol implemented using the Vertigo model to safely coordinate intersection crossings in the presence of omission failures.
- Chapter 6 shows the performance of the Vertigo implementation and the coordination protocol in an intersection management scenario and finds bounds on sensor and communication capabilities, and traffic rates under which the implementation can achieve progress.
- Chapter 7 concludes the thesis.

Chapter 2

Related Work

The Vertigo communication model addresses a need in cooperative automated driving for reliable communication in vehicular networks through a novel group communication model. This chapter discusses the state of the art in each area and the gaps left by existing solutions.

2.1 Cooperative Automated Driving

Cooperative automated driving is the use of communication between vehicles to facilitate and enhance automated driving through cooperation [Tsugawa 02]. An (uncooperative) automated vehicle might be capable of driving autonomously, but is limited in its world knowledge to what it can gather from its sensors, and in its actions to what it can confirm as safe without knowing the intentions of other vehicles. Cooperative automated vehicles do not have such limitations, since they communicate sensor data and intentions over a vehicular network, and coordinate their driving decisions using distributed algorithms.

Two of the most frequently studied cooperative driving scenarios are highways and intersections. Of key interest are the problems of merging onto a lane safely, joining and leaving platoons, and crossing a road safely. These are complex interactions between groups of vehicles, each of which needs to participate to ensure the overall safety of the system is guaranteed.

2.1.1 Automated Highway Systems

An Automated Highway System (AHS) comprises a set of technologies that together facilitate automated control of vehicles on a highway. The primary mode of driving on a highway is to follow preceding vehicles, keeping roughly the same speed. Human drivers are not very good at performing this task with global efficiency, because their view is limited to the vehicle directly ahead [Van den Broek 10]. It is often hard to tell whether a vehicle will accelerate or decelerate when no knowledge of its predecessors is available. This leads to poor decisions, which may result in ghost traffic jams [Ploeg 11], and necessitates greater distances between vehicles, leading to reduced highway capacity. Automated highway systems can overcome these problems by communicating the state and intentions of vehicles over greater distances at low latencies, and using the information as input into the control system.

The construct most commonly applied is that of a platoon. A platoon is a line of cars that stay within a certain distance of each other while driving and stay on the same lane. Platoons have been studied most prominently by the California PATH project [Shladover 08]. The project had several demonstrations of platoons on highways. Longitudal control (in the forward direction) is based on forward-looking radar and radio communication, while magnetic tracks are used to guide vehicles laterally. Communication is used to obtain reliable velocity and acceleration values from neighbours, such that the gaps between vehicles can be minimized. Communication failures are dealt with by increasing the gap, and falling back to radar as the primary source of distance information. Other work in the PATH project shows how vehicles can leave or join a platoon in the presence of other platoons [Horowitz 04]. The action is coordinated by platoon leaders, which instruct other vehicles in the platoon to choose a certain alignment. However, communication reliability is not addressed in this protocol, nor has it been demonstrated in practice. Another major platooning project is SARTRE, which demonstrated road trains in which only the first vehicle is driven manually and the other vehicles simply repeat its actions obtained from beacons [Chan 11]. Communication failures are detectable as missing beacons, but failures are ultimately dealt with by passing control back to a human driver.

Cooperative Adaptive Cruise Control (CACC) is a weaker notion of a platoon, but one that is possible using today's technologies [Bu 10]. CACC is an extension of Adaptive Cruise Control (ACC), which uses radar to adapt the cruise control speed of the vehicle to the speed of the vehicle in front. There is some delay before ACC software can reliably determine that the vehicle in front has started accelerating again after braking. Several rounds of radar measurements may be required to confirm the distance is growing. The effect of a single vehicle braking can therefore cause vehicles behind it to brake longer, which can perturbate and cause a traffic jam. CACC solves this problem by providing the cruise control software with recent information on the state and intentions of its predecessors through wireless communication. The controller can then know with a delay of at most one beaconing interval when vehicles in front of it start accelerating. CACC controllers can create a platoon that is string-stable [Ploeg 11], meaning the effects of acceleration and deceleration do not perturbate. CACC has been shown to work even if only part of the vehicles support it, and can also be benefial as a pure driver assistance system [Van den Broek 10]. There have also been demonstrations of CACC with vehicles and software from many different vendors in the Grand Cooperative Driving Challenge [Ploeg 12]. CACC is by far the most practical and mature solution to platooning, but it is also technically limited. It does not create a coherent structure which would allow merging operations or create highly compact platoons, and CACC controllers do not offer any safety guarantees. The driver is expected to intervene in worst-case scenarios.

A Cohort is a formalized notion of a platoon [Le Lann 11]. It requires a reliable neighbour-to-neighbour (N2N) channel that can be implemented in hardware through unidirectional antennas, accurate positioning, and forward-looking radar. The N2N channel combined with accurate positioning can provide the same function as the radar, providing functional redundancy and robustness to a single hardware failure. When a telemetry or communication failure occurs, cohorts can split to preserve the integrity of the cohort. A cohort imposes a geometric and network topological structure to a line of vehicles on a lane, but does not impose a particular coordination strategy. Instead, it provides a structure for implementing safe coordination algorithms using reliable communication primitives.

The cohort concept is complemented by a group structure for interactions between cohorts. Groups function like roles in a coordination scenario. Membership of a group is based on scenario-specific, ad-hoc conditions, such as receiving a message or deciding participation in the scenario based on position. Groups can be used for defining, reasoning about, and verifying communication protocols. The Zebra protocol suite presented in [Le Lann 12] supports lane changes in cohorts. The source vehicle and group of eligible receivers perform a 3-way handshake beginning with a selective (filtered) geocast, followed by unicast responses to the source, followed by a multicast by the source vehicle to the eligible receivers to announce a decision about a lane change. Replicating messages over the neighbour-to-neighbour links provides robustness to up to $\lfloor 2n/3 \rfloor$ omission failures for n receivers in bounded time. The time bounds are independent from the size of n, but an underlying assumption of the failure model is that broadcast failures are uncorrelated, which is not generally true in wireless networks.

2.1.2 Intersection Collision Avoidance

Intersections are typically managed either through stop signs, or traffic lights. The use of stop signs minimizes waiting time in low traffic conditions as cars can enter immediately or as soon as cars that were already there have passed. In high traffic conditions, greater throughput can be achieved using traffic lights, though most vehicles have to wait some time before being able to enter the intersection [Board 10]. Cooperative automated vehicles can potentially achieve both low waiting time and high throughput as the junction can be used more efficiently when vehicles coordinate their trajectories and know each other's intentions.

The problem of crossing an intersection without colliding with vehicles coming from different directions is most commonly referred to as Intersection Collision Avoidance (ICA). Existing automated vehicles are capable of crossing intersections safely [Levinson 11], but cooperative solutions can potentially achieve much better efficiency by planning trajectories in advance. [Lee 12] present the design of an intersection controller to which vehicles communicate that increases traffic throughput by 33% compared to a conventional control system, but perfect communication and 100% deployment are assumed. However, their efficiency results are comparable to the seminal work on Autonomous Intersection Management (AIM) by [Dresner 08], which does take into account failures. Vehicles anticipate messages from the junction controller and can detect their absence. The junction is divided into a 2-dimensional n by n grid, in which each grid cell represents a resource. Vehicles compete for exclusive, temporary access to the resources. Vehicles must obtain a contiguous set of resources from the lane on which they arrive to the destination lane in order to cross. The controller optimizes resource allocation for maximum efficiency. A drawback of AIM is that every intersection needs to be equipped with a controller (though it could be managed remotely over mobile networks), which becomes a single point-of-failure. There is no strong safety verification of the algorithms, and the number of safety incidents in simulations is greater than 0. Break-down of vehicles is dealt with by sending warning messages to other vehicles and the intersection manager, which is not robust to unbounded omission failure.

[Sin 11] present a fully distributed variant of AIM that relies on the capability offered by the Vertigo communication model (by forward reference) for its safety, namely geocast with reliable success detection. Vehicles that approach the junction send a message to an area around the junction to allocate a set of resources. If success is confirmed, the vehicle may proceed onto the junction. Formal analysis of the algorithm proves that it is safe and has no deadlock for a junction represented as a 1 by 1 grid [Asplund 12].

More fine-grained grids allow better efficiency, since it allows more vehicles to be on the road simultaneously, but complicates verification.

A real implementation of a distributed ICA system was demonstrated by [Milanes 11]. It involved multiple different autonomous vehicles driving around in an 8-loop. Vehicles share state information through periodic beacons. A local controller applies basic right-of-way rules based on the available state information, and only one vehicle is allowed on the junction at a time. No specific measures are taken to deal with communication failures and the safety of the algorithms is not verified. However, it is noteworthy since it is one of the first real-world demonstrations of cooperative ICA.

2.2 Vehicular Networks

The idea of vehicular communication is very old, with regular engineering publications appearing as early as 1952 [IRE 52]. Clearly, these were targeted at analogue radio, not digital communication networks, but it stresses the importance of the field. More famously, the 1958 Disney cartoon "Magic Highway U.S.A." [Clemmons 58] showed a vision of vehicles communicating in order to automate and modernize driving. Today their vision is still far-fetched, but getting a little closer every day.

The field of vehicular networks has grown enormously in recent years and has been subject of many recent surveys [Panichpapiboon 12,Karagiannis 11,Trivedi 11,Willke 09]. The surveys cover a variety of issues such as transmission power control, channel access, routing techniques, multicasting, security and different communication technologies. The field is already far too broad to describe it appropriately here. One particular survey classifies our work quite well. Willke et al. show the relationship between several classes of vehicular applications and some of the current work in vehicular networks [Willke 09]. The four application classes are:

1. General information service

2. Information services for vehicle safety

3. Individual motion control using inter-vehicle communication

4. Group motion control using inter-vehicle communication

Vertigo specifically targets the 4^{th} class of applications, referred to here as cooperative automated driving, which includes automated highway systems and intersection collision avoidance. The survey discusses the communication requirements of each class with respect to several topics. Type-4 applications require deterministic reliability, meaning they need the ability to determine whether information was received. In terms of scalability and scope each vehicle is expected to communicate with on the order of 10-100 other vehicles in a relatively small region. Another important property of type-4 applications is the need for a persistent group structure. Vehicles need to explicitly register their membership to share data, and understand roles and responsibilities. In terms of latency there are some type-4 applications that only degrade when messages are delivered after a deadline (soft real-time) and others that fail (hard real-time).

2.3 Geographic Communication in Vehicular Networks

For the purpose of cooperative automated driving, one of the most relevant vehicular networking concepts is that of geocast. The term geocast was first coined by [Navas 97] to describe broadcast of a message in an area defined by one or more circles or polygons in the GPS coordinate system. Their approach mainly considered infrastructural networks in which the benefits of geographic addressing are limited, but not much later the concept was applied to mobile ad-hoc networks (MANET) in which geographic addressing simplifies routing and maps well to mobile applications. Location Aided Routing (LAR [Ko 98]) uses data from GPS to find network routes in a MANET. LAR is not a true geocast protocol as it only provides unicast, but is very similar in its workings. The protocol assumes the source node has some information about where the destination node resided at time t_0. Since t_0 is in the past, the source needs to compute the *expected* *zone*, which is the zone the destination node could be in currently. It then decides on a *request zone*, which should contain the expected zone, and attaches it to the outgoing message. If the source is outside of the request zone the message needs to be forwarded to one or more nodes in the zone. Intermediate nodes decide whether to forward the message, depending on whether they are closer to the destination position (at t_0). When the request zone is reached, the message is forwarded (flooded) by all nodes whose GPS shows that they are present in the zone. The purpose of LAR is to establish unicast routes. Nodes keep track of the source of the message and the destination sends back a route reply via the first route over which a message arrives. In a follow-up paper the authors showed that the technique can be applied equally well to broadcast or multicast (filtered broadcast) to all nodes in an area [Ko 99]. In this case, the position of the destination is replaced by the geographic centre of the request zone and no route establishment is necessary. The concepts of a request zone and expected zone appear in a generalized form in the Vertigo communication model.

A comprehensive survey of geocast techniques in vehicular networks is provided in [Allal 12]. One of the challenges in vehicular networks is the highly variable node density. When node density is low, the network may partition and there may not be any routes to the destination. When node density is high, flooding can lead to a large number of collisions on the channel if forwarding is not managed correctly. This last phenomena is more widely known as the broadcast storm problem [Tonguz 06]. Since geocast often involves flooding within the target area, the same problem occurs. An early approach to adapt geocast to the node density in vehicular environment is Inter-Vehicle Geocast (IVG [Bachir 03]). The protocol exploits the fact that messages only need to be forwarded along the direction of the road. Vehicles defer forwarding of a message for a short period and cancel the transmission if a vehicle further down the road broadcasts it. To deal with situations where node density is low, IVG rebroadcasts the message periodically at an interval that allows vehicles to react to the message before they come into braking distance. In IVG, the source is assumed to be static and ever present. More recently, it was found that IVG still has a more localized form of the broadcast storm problem, caused mainly by a lack of synchronization of wait times [Ibrahim 09]. The authors proposed an extension named p-IVG, which only starts the rebroadcast timer with probability $1/density$ and otherwise stops forwarding immediately.

In a high-paced environment in which low-latency networking is not always possible, sending a message to an area becomes ambiguous. It is not clear to what point in time or time frame the area refers. The situation can change significantly in between the moment of sending, first reception, last reception and potentially first response and last response. An approach to deal with this problem is a stored or *time stable* geocast [Maihofer 03]. In this form of geocast, the message is meant to be delivered to every car that is within a delivery area before the expiration of the geocast, including those that are not yet there when the geocast is started. Although functionally beneficial, the persistence is difficult to implement. The authors suggest assigning or electing a server that stores and repetitively geocasts the messages to the delivery area. An alternative is to have all nodes store messages for their current location, and exchange them when a new neighbour enters one of the delivery areas. These techniques fail when density becomes too low, but more recent work has addressed these problems by anticipating failures. Adapting the size of the area in which messages are forwarded to the expected density can significantly improve the probability of the message being delivered once the vehicle is actually in the delivery area [Hermann 07]. In a vehicular network this could be achieved by utilizing vehicles travelling in the opposite direction [Yu 08].

Geocast and geographic routing in vehicular networks is standardized by GeoNet [Tsukada 10], which ties the target area of a geocast to an IPv6 multicast address in order to be compatible with existing and future IPv6 applications. GeoNet does not specify the routing protocol. A large number of suitable routing protocols are explored by [Taysi 12], but no clear winner exists due to various different trade-offs that are made in each protocol, which may suit some applications better than others. The specification of the area is also still a subject of study [Jochle 12], and no standardized delivery area formats exist as of yet.

2.3.1 Reliable Communication in Vehicular Networks

This thesis presents a group communication system for reliable, real-time communication. However, "reliable" can be an ambiguous term, describing different, yet related concepts. Further confusion is caused by its relationship to the term "reliability", which is typically used as an abstract quantity.

One meaning of the term "reliability" is the probability of successful communication. In this case, "reliable" would refer to a probability of 1, but this is not achievable in a vehicular network. The density in a vehicular network is highly variable causing it to suffer from contention in high density conditions, and partitioning in low density conditions and obstructed environments. The DV-Cast protocol improves delivery probability by combining broadcast suppression (as applied in IVG), with store-carry-forward to overcome gaps [Tonguz 10]. While such reliability is very useful and can improve application-level performance, it does not offer any guarantees for time-bounded applications.

Given the time constraints of vehicular networks, the probability of successful communication depends heavily on the available time window. Bai et al. [Bai 07] suggest to use the *T-window reliability* metric, defined as the probability of successfully receiving at least one packet from a neighbour during a time window T. The authors target a type of beaconing application where only a single beacon per car per time window is needed, but the principle can be equally well applied at a per-packet basis for more complex applications. The recommendation is that communication system developers should express their reliability as a function of the time window, such that application developers can use this to optimize their time-constrained data transfer.

Given a lossy channel, an application can correctly determine whether communication succeeded through positive acknowledgements. The implementation is eventually "reliable" through retransmissions and by addressing the specific challenges of vehicular networks, such as the highly variable node density. For example, in the AckPBSM broadcasting protocol by [Ros 09], nodes store and keep forwarding a message until all their neighbours have acknowledged it. This significantly reduces the likelihood of messages being lost due to individual collisions or omission failures, but still does not give guarantees within a finite time window.

This thesis uses the term "reliable" primarily to describe the determinism offered through communication feedback in hindsight. The ability to distinguish success from failure allows applications to reason about communication in a way that is *reliable*. This refers to the reliability of information (on success), rather than communication, and effectively sits above the network layer.

2.4 Group Communication Systems

Vehicular networks provide the basic communication infrastructure for disseminating messages. However, cooperative automated vehicles may require the ability to reason about the outcome of communication, such that they can reason about the behaviour of other vehicles, which can be provided by an additional middleware layer. A view-oriented group communication system (GCS) provides communication and membership services in support of multi-point to multi-point (N to N) communication. The main purpose of a GCS is to take away the complexity of providing reliable, distributed communication primitives from the application.

A GCS is usually tailored to a specific application domain, such as distributed databases, clustered operating systems or highly-available servers. Most work in GCSs happened in the 1990s and early 2000s. A comprehensive survey of view-oriented group communication and a formal definition was given by Chockler [Chockler 01]. Since then, group communication has attracted less attention. This may be due to the increased attention to large-scale systems in which complete membership is infeasible and more focus is on asymmetric (N to M) architectures.

2.4.1 Real-time and Mobile Group Communication

Real-time group communication is challenging due to the fact that strict time constraints preclude reliability guarantees. [Kim 99] gives several definitions regarding success and completion of real-time, fault-tolerant multicast. The multicast is *successful* if and only if the sender receives an ACK from every receiver within the *acknowledgement time bound.* The multicast is *complete* if the last acknowledgement arrived, a failure is detected or the acknowledgement time bound expires. The same methodology is used in the Vertigo communication model.

The usefulness of real-time group communication in mobile coordination applications was recognized by Nett and Schemmer [Nett 03]. They propose to use the access point in an 802.11 network as a centralized group communication system. This allows reasonably simple implementations of membership, atomic multicast and total ordering in a timely fashion. Unfortunately, their approach relies on the presence of a synchronous communication channel that is both timely and reliable, or has at most a known number of omission failures within the time bound. Although this may be feasible in a static environment where the channel is predictable and safety is not absolutely critical, it is not applicable to the vehicular environment.

In mobile ad-hoc networks, the challenge of providing membership in an ever changing environment has also been recognized. The Vertigo model exploits the maximum speed of vehicles to reason about membership changes over time. Roman et al. took a similar approach in their consistent membership service [Roman 01]. Similar to Nett and Schemmer, their protocol assumes real-time communication to be reliable if the receiver is within a fixed communication range. Vehicles are required to be in the same group only if they are in a safe distance of each other. The safe distance is defined in such a way that even if vehicles move away from each other at the maximum speed, they will still be on the edge of the communication range. If the assumptions hold, the system allows views to be maintained reliably for a known period. However, the reliability assumption ignores obstacles, background traffic and atmospheric effects that can make communication, especially when restricted by time, unreliable even at short distances.

2.4.2 Reliable Group Communication

One strategy to dealing with unreliability in vehicular network is to accept it and provide insight into probability. Route Driven Gossip (RDG) [Luo 04] is a protocol that aims to provide reliable multicast with predictable probability. The system maintains a view of the membership group, but does not attempt to confirm reception or guarantee delivery. Instead, it predicts the probability that a fraction of the messages will arrive at a member. Applications can adapt to the probability values, by either taking measures that increase the probability or adapting behaviour if the probability is insufficient. Unfortunately, RDF is not quite suitable for vehicular networks or safety applications. One problem is that metrics like packet loss that RDG uses to predict probability are themselves much harder to predict in a vehicular network. A car may suddenly disappear behind an obstruction that blocks communication. When safety is concerned, even very high probability values that are associated with life-critical systems would be insufficient unless it is also highly certain that the value is accurate. In general, safety-critical systems should not treat events as probabilistic, but should instead ensure any known sequence of events that could lead to failure does not occur.

The space-elastic communication model [Bouroche 06a] is a predecessor of the Vertigo model. It uses feedback to determine the space in which a message can be delivered in real-time. The model relies on relatively low mobility and does not consider the decay of the area or the set of potential receivers. SEAR [Hughes 06], the implementation of the model, assumes reliable delivery of negative acknowledgements to provide adaptation notification and the reliable delivery of messages to nodes that are in an area for a sufficiently long time. Nonetheless, it takes a more realistic view than to simply assume that communication always succeeds.

An important related work addressing deterministic reliability is the Reliable Neighborcast Protocol (RNP [Maxemchuk 07]). It acts as an overlay on top of a 1-hop reliable broadcast protocol, M-RBP [Willke 05], to make it suitable for highly mobile vehicles. M-RBP uses a token ring protocol, in which receivers take turns in sending a control message containing acknowledgements at fixed time intervals. Senders listen for the round immediately following a transmission to confirm whether all nodes received it and which are missing. If one of the members of the ring does not receive a control message at the scheduled time it starts a distributed voting procedure. If a majority of ring members did not receive the message, the node is removed from the ring. Otherwise, the receivers know which other receivers should have received the control message. Uncertainty can arise if failures occur in the voting procedure and no majority can be determined, but this is detectable. The problem with using M-RBP in a vehicular environment is that vehicles constantly move out of the broadcast group. RNP provides an overlay which guarantees that a vehicle is in at least one and at most two overlapping groups. It defines a set of rules that vehicles should follow to determine when they should join and leave a group. The main difference in terms of guarantees offered by RNP vs. Vertigo is that RNP simply removes vehicles from the group when communication fails. Additionally, vehicles may not be aware of existing groups in an area. Under extreme circumstances, this could lead to partitions that overlap spatially and cause vehicles to falsely conclude that communication to neighbours succeeded.

2.5 Discussion

Each of the reliable communication solutions discussed offers some form of reliability, but to some extent fails to offer a rigorous, safety-critical guarantee. Reliable routing protocols such as AckPBSM and DV-Cast fail to offer guarantees under time constraints. A protocol such as RDG which can estimate probability may be useful, but it is difficult to see it being used for safety-critical scenarios where probabilities less than or distinguishable from 1 are unacceptable. SEAR assumes that communication is reliable after the initial scheduling phase, while Nett and Schemmer assume a bounded number of omission failures within a time window, neither of which are realistic. RNP can detect communication failures within a bounded time in a communication group, but multiple communication groups can exist that overlap physically. Cohorts also define a reliable communication scheme, which does seem feasible by creating neighbour-to-neighbour communication channels using unidirectional antennas that do not suffer from collisions or obstacles and only need to communicate over a short distance. If a neighbour-to-neighbour channel fails, this can be immediately detected by the absence of a beacon while ranging sensors show a vehicle is near. However, neighbour-to-neighbour channels only provide communication within a cohort on a single lane. The Zebra protocol suite provides reliable bounded-time cohort-to-cohort communication, but relies on a failure model which assumes that omission failures in a wireless network are bounded and uncorrelated.

While none of the communication protocols provide a solution that can truly be relied upon by a safety-critical application, cooperative automated vehicles have a strong need for it. Intersection collision avoidance scenarios may have obstacles blocking communication and causing omission failures. Vehicles may not notice each other at all on the communication channel. Cooperative automated vehicles on highways suffer from variable density, which gives rise to both high contention and a high degree of partitioning on the wireless network, causing similar omission failures. This could be particularly problematic in lateral coordination scenarios such as lane changes and on-ramp merging. The remainder of this thesis presents the Vertigo communication model, which provides a geocast operation with reliable success confirmation, which addresses the shortcomings of existing communication solutions in fulfilling the requirements of cooperative automated driving solutions.

Chapter 3

System Model

This chapter introduces models for membership and group communication for cooperative automated driving applications. Together these form the *Vertigo Communication Model* comprised of a set of interfaces and data structures, their semantics, and the guarantees offered by them. A reference implementation of the model is presented in the next chapter. The model is designed to be both feasible to implement under reasonable assumptions, and beneficial to applications. To clarify the rationale for the models, this chapter first describes the design principles and requirements of cooperative automated driving that led to the current definition. The requirements follow from the overall goal of being able to confirm the safety of an intended manoeuvre in a distributed system of vehicles. The problem that systems implementing the Vertigo model solve is to deliver a message to all vehicles that may be in a given area at a given time and, if no omission failures occur, provide a reliable confirmation of success.

3.1 Design principles

The Vertigo model is partly based on a set of design principles that follow from existing work [Bouroche 06b]. They are explicitly listed here in order to narrow the design space of the model. Alternative approaches that do not follow these principles may be viable, but are beyond the scope of this dissertation.

Table 3.1: Design principles

Geographic group communication	From the perspective of a vehicle in a cooperative automated driving system, interactions typically involve a set of vehicles within a surrounding *area*. Communication between the vehicles constitutes a form of *many-to-many* or *group communication*, with partially overlapping groups. Without prior knowledge of each of the vehicles in the area, the groups can only be addressed *geographically* for the purpose of communicating with them.
Real-time control	Driving decisions have stringent *real-time constraints*. A communication system that supports an application making driving decisions must allow the application to act on the outcome of communication within *bounded time*.
Adaptation	Failures of the communication channel cannot be prevented. Therefore, no communication system can guarantee that its messages *will* arrive within a bounded time. The Vertigo model exploits the fact that it is possible to confirm whether a message has arrived in hindsight through acknowledgements. However, a sender cannot distinguish between an omission failure of a message or its acknowledgement. Applications need to deal with this reality and have a way to safely adapt to the possible failure scenarios that could have occurred in the absence of confirmation. An example would be to stop the vehicle if the safety of going forward cannot be confirmed.

Safety-critical	The system only reports success with safety-critical certainty, or reports that it cannot confirm success. In other words, there is no probabilistic or best-effort notion of success.
Decentralized	A decentralized solution is preferable, since a centralized solutions would create a single point-of-failure for road traffic. Moreover, communication failures between the central controller and the vehicle may occur. The central controller cannot rely on its control decisions being acted upon, nor can the vehicles it controls. Vehicles can only rely on locally available information to take safe driving decisions, which means the problem is inherently decentralized.

3.2 Cooperative Automated Driving Requirements

Cooperative automated driving requires planning algorithms with strict safety and timeliness guarantees. Vehicles must at all times preserve a set of safety constraints and anticipate worst-case scenarios to know that the actions that they take will not violate any safety constraints in the future. This section will use a formal model of driving to show that the need for the membership and group communication models presented in the next sections follows from the safety requirements. The solution presented in this section is not necessarily the only way to do safe, cooperative automated driving, and also not the only solution to which the membership and group communication models or their characteristics can be applied. The solution is presented here to motivate the design of the models, which ultimately follows from safety constraints.

3.2.1 Safety constraints

Safety constraints on vehicles can be defined as logical propositions. Given the global set of vehicles $\mathcal{V}$ and the position of a vehicle $i \in \mathcal{V}$ at time $t \in \mathbb{R}^+$: $P(i,t)$, and a **distance** function, one can define the safety constraint as the assurance that the distance between vehicles is always greater than 0 (no crashes): $\forall t \in \mathbb{R}+, i, j \in \mathcal{V}$:

$\textbf{distance}(P(i,t), P(j,t)) > 0$. This safety constraint must always hold for any automated driving system. Further constraints are necessary to create a safe system, particularly to allow the safety constraint to be preserved given finite deceleration. For the sake of simplicity and genericity, only the implications of the safety constraint itself are considered in this chapter.

We take an absolutist view on safety. Before taking any action, a vehicle must ensure that it will not enter into a state in which the safety constraint might be violated in a future worst-case scenario. There must at all times be a way for a vehicle to transition to a state in which no safety-constraints are broken. A state is deemed safe if this is the case, and unsafe otherwise.

3.2.2 Speed and locality

A constraint on the speed of a vehicle i at time t: $V(i,t)$ can be defined, namely that $\forall t \in \mathbb{R}, i \in \mathcal{V} : 0 \leq V(i,t) \leq v_{max}$ where $v_{max} \in \mathbb{R}^+$. In other words, vehicles cannot go faster than v_{max}. For completeness, speed can be defined as a step function with $\forall t, \delta \in \mathbb{R}^+, i \in \mathcal{V} : V(i,t) \Rightarrow (\textbf{distance}(P(i,t), P(i,t+\delta)) = \delta \times V(i,t))$, for arbitrarily small time steps of length δ.

While trivial, the definition of a speed and maximum speed has important implications. In particular, it bounds the set vehicles with which the safety constraint can be broken within a finite time frame. Given a future time t_{future} and the current time t_{now}, then the only vehicles j which could violate the safety constraint with a vehicle i are those for which $\textbf{distance}(P(i,t_{now}), P(j,t_{now})) \leq (t_{future} - t_{now}) \times v_{max}$ holds.

3.2.3 Actions

Because computation and communication are inherently discrete functions, software in control of a vehicle must, by definition, make decisions for a future time frame in which the vehicle will be in a certain area. Driving can be modelled as a vehicle i taking an action with constraint $Q(i, A, t_{start}, t_{end})$, which takes place in an area A and a time frame between $t_{start} \in \mathbb{R}^+$ and $t_{end} \in \mathbb{R}^+$ with $t_{start} < t_{end}$. When it comes to actions, we define that it must be the case that $\forall t \in \mathbb{R}^+, i \in \mathcal{V} : Q(i, A, t_{start}, t_{end}) \Leftrightarrow ((t_{start} \leq t \leq t_{end}) \Rightarrow P(i,t) \in A)$. In other words, the vehicle does not leave A during the time frame. Furthermore, an action is only safe if all states that can be reached during the time frame by performing the action are safe. Note that this definition does not put any constraints on the vehicle's behaviour or the software, it is merely a symbolic definition to express and reason about continuous transitions.

3.2.4 Automated driving

The problem of automated driving can now be expressed as finding actions that allow progress and are safe. Section 3.2.2 established that there is a bounded set of vehicles that could violate the safety constraints by end time t_{end} due to the maximum speed. Each of these vehicles is within an area A' which includes all positions that lie less than $(t_{end} - t_{now}) \times v_{max}$ away from the action area A at current time t_{now}. The safety constraint is known to hold for vehicles outside A', and therefore the safety of an action only needs to be confirmed in relationship to vehicles that are inside A' at t_{now}. Phrased differently, safety only needs to be confirmed for vehicles that *could be* in A by t_{end}, given their maximum speed v_{max}.

It is not necessarily the case that vehicles outside of A' are not in any way involved in confirming whether a potential action is safe. This relates primarily to the constraints that are not specified. For example, a vehicle j inside A' might need to stop for an action Q to be safe, but whether j can stop may depend on vehicles outside A'. This makes the problem of automated driving very difficult to solve if vehicles need to confirm the safety of an action solitarily. However, cooperative driving can make the problem simpler.

3.2.5 Cooperative automated driving

If vehicles have the ability to communicate, the way in which the safety of an action is determined can be simplified. Let us say that every vehicle j can determine whether its own intentions comply with a candidate action $Q(i, A, t_{start}, t_{end})$ desired by vehicle i. In other words, i proceeding with Q does not lead to a violation of the safety constraint with j. In that case, a viable solution to determining the safety of an action is if vehicle i communicates Q to all vehicles that *could* be in A by time t_{end}.

Every receiver of Q from i determines whether Q complies with its own intentions, and responds to i to confirm or reject the action. If i can confirm it received a positive response from all the vehicles that could be in A by time t_{end}, then the safety of the action is confirmed. However, due to communication unreliability and the mobility of vehicles, responses may be omitted and the set of vehicles that could be in area A at time t_{end} may not be known. A candidate action that requires confirmation from other vehicles can only be taken if successful communication can be confirmed. Otherwise, an alternative course of action should be taken that is known to be safe.

Cooperative automated driving as presented here requires the ability to communicate to the set of vehicles in a specified area at a specified time and a reliable confirmation of successful delivery to every of the vehicles in the set. The Vertigo communication model presented in the remainder of this chapter fulfills this requirement.

3.3 Membership model

This section describes the membership model used in the group communication system, which is one of the core contributions of the thesis. The membership model defines operations and constraints on *membership tuples*. The tuples are pieces of information with a small calculus consisting of decay and merge operations. These operations allow tuples to be combined across large areas and different points in time, while maintaining a guarantee that is sufficient to fulfill the requirement of knowing all vehicles in an area.

3.3.1 Membership tuples

A membership tuple is a relationship describing the potential presence of vehicles in an area at a given time. A tuple holds three data elements: a set of (network) identifiers of vehicles M, an area A, and a time t, written $\mathcal{M}(M, A, t)$. The information conveyed in the tuple is expressed as a constraint on the tuple known as the membership constraint: If a vehicle i is in area A at time t, then its unique identifier i must be in M. A formal definition is given below.

$$\forall i \in \mathcal{V}, t \in \mathbb{R}^+ : \mathcal{M}(M, A, t) \Leftrightarrow (P(i, t) \in A \text{ at } t \Rightarrow i \in M).$$

A membership tuple may only be exposed to an application or shared with other vehicles if the constraint is known to be true. Membership tuples can be generated using sensors, or obtained over the network from infrastructure or other vehicles. It is expected that any component providing a membership tuple has ensured the membership constraint holds. Given this guarantee, membership tuples can be used to reason about the states of vehicles. Whatever holds true for all vehicles whose identifier is in M holds true for all vehicles in area A at time t. Additionally, the absence of a vehicle from area A at time t can be reliably confirmed if its identifier is not in M.

It is worth noting that the membership constraint has a one-way implication and therefore does not necessarily convey information about vehicles whose identifier is in M. A tuple $\mathcal{M}(M, A, t)$ implies that vehicles not in M are not in A at time t, but vehicles whose identifier is in M could be anywhere. It is generally expected that vehicles whose identifier is in M are also in A, but M may be a superset of those vehicles.

3.3.2 Generating Tuples

There is no pre-defined way in which tuples must be generated. Tuples can be derived from local knowledge of the environment. For example, if vehicle i knows from its sensors that an area A around vehicle i is void of any other vehicles at time t, then it can derive a tuple $\mathcal{M}(\{i\}, A, t)$. Ranging sensors measure empty space with high reliability. It is extremely unlikely that a fine-grained ranging sensor will not observe any reflections from an object the size of a vehicle and thus falsely measure it as empty space [Moras 10]. In addition to generating tuples from local knowledge, tuples can be derived from other tuples using the *decay* and *merge* operations. They can also be serialized and communicated over the wireless network, such that tuples can be cooperatively constructed.

3.3.3 Decaying tuples

Membership tuples are bound to a specific point in time t. However, when a tuple is obtained, it may be bound to a different point in time than the point in time in which the application is interested. Fortunately, a tuple that is bound to a time that lies in the future relative to t can be derived from the tuple based on restrictions on the behaviour of vehicles. In particular, vehicles cannot go faster than some maximum speed v_{max}, which may be set high enough to ensure it is never reached by ordinary vehicles. Restrictions can be defined as a property of elements of the road network.

If it is known that there are no other vehicles than those whose identifiers is in M in area A at time t, then it also known that there exists some area smaller than A into which no other vehicles than those in M could have entered before $t + \delta$, even if they were driving at speed v_{max} from the boundary of the area. This derivation is made using the *decay* operation on membership tuples, which shifts time t forward by δ, but shrinks the size of area A by a distance $\geq v_{max} \times \delta$. The value of v_{max} may be dependent on location and the specific shape of the area, which is implementation-dependent. An implementation of the **shift** function to shrink the area is defined in the next chapter. Given such a **shift** function, decay can be defined as follows:

```
function $\textbf{decay}(\mathcal{M}(M, A, t), t_{decayed})$:
$\textbf{return}\ \ \mathcal{M}(M, \textbf{shift}(A, t_{decayed} - t_{now}), t_{decayed})$.
```

3.3.4 Merging tuples

The membership constraint permits a logical union of membership tuples to be performed. For a pair of tuples $\mathcal{M}(M_1, A_1, t)$, $\mathcal{M}(M_2, A_2, t)$ with the same timestamp t, it follows directly from the constraint that for any vehicle that is in $A_1 \cup A_2$, its identifier is in $M_1 \cup M_2$. This means that a new tuple $\mathcal{M}(M_1 \cup M_2, A_1 \cup A_2, t)$ can be derived. Tuples with different timestamps can be merged by first applying decay using the **shift** operation. The algorithm for merging arbitrary tuples is as follows.

```
function $\mathcal{M}(M_1, A_1, t_1) + \mathcal{M}(M_2, A_2, t_2)$:
$t_{merged} \leftarrow \mathbf{max}(t_1, t_2)$.
$A_{merged} \leftarrow \mathbf{shift}(A_1, t_{merged} - t_1) \cup \mathbf{shift}(A_2, t_{merged} - t_2)$.
return $\mathcal{M}(M_1 \cup M_2, A_{merged}, t_{merged})$.
```

When merging a set of membership tuples, the oldest tuple must be repetitively decayed to the timestamp of the second oldest tuple, to lose as little information as possible when performing decay. An algorithm for merging a set of tuples is given below.

```
function $\mathbf{merge}(tuples)$:
$tuples \leftarrow \mathbf{sort}(tuples$ in reverse chronological order)
for $i = |tuples| - 2 \rightarrow 0$ do
    $\mathcal{M}(M_i, A_i, t_i) \leftarrow tuples[i]$.
    $\mathcal{M}(M_{i+1}, A_{i+1}, t_{i+1}) \leftarrow tuples[i+1]$.
    $A'_{i+1} \leftarrow \mathbf{shift}(A_{i+1}, t_i - t_{i+1})$.
    $tuples[i] \leftarrow \mathcal{M}(M_i \cup M_{i+1}, A_i \cup A'_{i+1}, t_i)$.
end for
return $tuples[0]$
```

3.3.5 Discussion

A membership tuple provides a binding between the physical presence of vehicles and their logical presence on the network. For a given membership tuple, it is guaranteed that what holds true for all vehicles whose (network) identifier is in M must hold true for all vehicles in A at time t. An application can use a membership tuple to confirm that all vehicles in a given area at a given time have received its message.

A beneficial property of the membership tuple model is that (un)certainty is weaved into the information a membership tuple conveys. As time progresses, the certainty implicitly shrinks, but if more information becomes available, the certainty grows. This is made explicit through the decay and merge operations. Membership tuples are naturally tolerant to changes in the physical world by specifying only the potential presence of vehicles in an area, rather than providing specific information on their location. These properties of tuples make it feasible to implement the membership model in such a way that useful, valid tuples can be constructed despite challenging conditions, as the following chapters will demonstrate. At the same time, membership tuples provide a reliable guarantee, which maps well to the requirements presented in Section 3.2. A vehicle must ensure that the safety constraint holds for any action it takes. The set of vehicles for which the safety constraint needs to be tested is bounded by the set of vehicles that could be in the area A to which the action is constrained by end time t_{end}. The membership model can be used to establish a superset of these vehicles (M), and to confirm safety for the set M is to confirm safety for all the vehicles.

3.4 Group Communication Model

The group communication model defines a set of operations to be implemented by a group communication system. The geocast operation sends a message to all vehicles that are in a specified target area at a specified target time. Vehicles can receive and respond to this message, and responses can be collected by the sender. The geocast operation is followed by a result event, which specifies whether the geocast was successful, meaning that all vehicles in the target area have received the message. The remainder of this section will describe and substantiate the model, and formally specify it.

3.4.1 Geocast

The main operation of the group communication model is a **geocast** primitive. Geocast aims to send a message to all vehicles in a particular geographic area. One of the benefits of geocast in a wireless ad-hoc network is that it does not require upfront knowledge of the presence of vehicles in an area. To perform a geocast, a vehicle broadcasts a message containing a delivery area over the wireless ad-hoc network. Other vehicles that **receive** the message check whether they are inside the delivery area based on their latest position information and if so, deliver the message to applications, and potentially forward the message to other vehicles in the area. With some probability, all vehicles in the area will receive the message before a certain time, though omission failures may occur and network partitions may exist within the area.

For the purposes of cooperative automated driving, the benefit of geocast lies in the fact that it maps well to the spatial nature of the problem. Driving actions, as modelled in Section 3.2.3, take place in a specific area, and all vehicles involved in the action from a safety perspective are in the surrounding area. The set of vehicles in the area is not known upfront, since it constantly changes. Therefore, geocast is a suitable communication primitive for cooperative automated vehicles to communicate about driving actions. Vehicles typically coordinate their behaviour with the vehicles in the area surrounding them. Interactions that a vehicle i has with vehicles in remote area A, will typically also involve the vehicles in i's present surroundings, since they can reach A at the same time as i. While some geocast models may target arbitrary geographic areas, the Vertigo model is limited to areas in which the sender of the geocast resides.

It may sometimes be the case that a message is only meant for a particular subset of vehicles in an area, such as those on a particular lane, but this level of filtering is best left to higher layers where more information on the state of the receiving vehicle is available. However, the model does define a basic filtering mechanism based on port numbers, a common way of isolating applications in network protocols. When the group communication system receives a message for a destination port to which no application is bound, it responds with a message to avow that it is *uninterested*. If an application is bound to the destination port, then the message is provided to the application and the application is given the ability to respond with a message of its own. No size limitation is specified, but it is imperative that responses are small, since there may be many of them. A receiver can elect not to respond, which the sender will treat as an omission failure. Responses are routed back to the sender. The sender can **collect** and read the responses from both interested and uninterested receivers as they come in.

3.4.2 Timeliness

To perform a geocast using the Vertigo model, an application needs to specify a number of points in time. The communication system needs to know to which point in time the target area of a geocast refers in order to decide which vehicles are eligible, it needs to know when to deliver a messages to the receiver(s), and when results need to be delivered to the sender. Each of these points in time are specified by the sender and included in the geocast messages, such that all receivers are aware of them.

While relying on synchronized clocks is a poor design choice in an 'ordinary' distributed system where clocks can be highly inconsistent, automated vehicles can be expected to have GPS, which provides highly accurate time. This provides vehicles with the ability to use the clock to organize sequences of events in a distributed system with only minimal coordination between nodes. A similar approach has been used to build a globally distributed, partition-tolerant database [Corbett 12]. Nodes can execute events almost simultaneously using only their clocks, given an execution time. Every node keeps a priority queue of events in the order of execution time and executes the first event in the queue when the clock is greater than or equal to its execution time. Even in the presence of clock jumps, nodes can guarantee a globally consistent ordering of events by never executing an event that was meant for an earlier time than the last executed event. The drawback of this approach is that some events may not be executed. However, this is an expected problem when transferring messages over an unreliable network. There is no guarantee of delivery.

The previously described mechanism can be used to give messages a globally consistent delivery time and ordering. The sender of a geocast message can specify an optional delivery time $t_{delivery}$, which is the time at which the message is to be delivered. If two senders specify the same delivery time, their unique identifiers are used to determine the order in which the events are executed. If a message with a delivery time in the past is received, it is discarded. If the delivery time lies in the future, the message is added to the delivery queue and delivered to the receiving application at the specified time. If no delivery time is specified the message is delivered immediately upon reception and no ordering guarantee is provided.

At some point after initiating the geocast, the sender will inspect the results and base a decision on them. The sender specifies the point in time at which the results should be made available as t_{result}. It is necessary that $t_{result} > t_{delivery}$, such that there is time to gather results after delivery.

Due to the mobility of vehicles, the set of vehicles in a target area changes over time. For this purpose, the application needs to provide a target time t_{target}. The geocast aims to deliver the message to all vehicles that are in the area at the target time. This target time needs to lie in the future and must be $\geq t_{result}$, since the system would otherwise have to keep track of which vehicles have visited an area in the past (before t_{result}). However, if the target time lies in the future, it is not yet certain for some vehicles whether they will be in the area at the target time. Likewise, it is not yet certain which of the vehicles currently contained in the area will be there in the future.

To ensure that the geocast is able to reach all of the vehicles in the set, the protocol needs to aim to deliver it to all vehicles that *might* be in the area in the target time. An implementation can achieve this by taking into account 'worst-case' behaviour by expanding the target area by $v_{max} \times (t_{target} - t_{now})$ to obtain the *delivery area* of the geocast. Effectively, the expansion of a target area to a delivery area anticipates the effect of decay and uses the inverse of the **shift** function used to implement decay.

3.4.3 Feedback with membership

The geocast operation is successful if it is known by the sender at t_{result} that all vehicles that might be in the target area at t_{target} received the message before $t_{deliver}$ (if applicable) and responded. The group communication system can provide this guarantee to the application at t_{result} under a specific condition. The **result** event at t_{result} for a corresponding **geocast** event contains a membership view and a set of interested responders $R_{interested}$ and a set of uninterested receivers $R_{uninterested}$, and the combined set is $R = R_{interested} \cup R_{uninterested}$.

Success is achieved if and only if, given a **geocast** with a target area of A_{target} and a target time of t_{target}, a membership tuple $\mathcal{M}(M, A, t_{target})$ can be found for which the following holds:

$$R \supseteq M \wedge A \supseteq A_{target}$$

In other words, all members have responded and the membership area covers the target area. This means that, even if some responders might not be members and some members might not be in the target area at the target time, all vehicles that will be in the target area at the target time have responded.

3.4.4 Sender API

When the sender initiates a **geocast**, it needs to specify the message data, the timeliness constraints of the geocast, and the area in which the message is to be delivered. Forwarding and delivery will occur based on those parameters and trigger a series of **collect** events to deliver any data that the receivers have sent back, followed by a **result** that can be used to check whether the geocast was successful. The notation from Appendix A is used to specify the API as follows.

Table 3.2: Sender API specifications

input	**geocast**($id \in \mathrm{ID}_{request}$, $m \in$ Message, $A_{target} \in$ Area, $p \in$ Port, $T \in$ Timeliness) at $t_{geocast}$.
description	The geocast event triggers forwarding of the data in m to vehicles in target area A_{target} at time t_{target} (part of T), to be delivered at applications that are listening on port p. T defines time constraints for the geocast. To identify the geocast, the application must specify a locally unique identifier, which will be used to associate events with the geocast. As a result of the geocast event, 0 or more **collect** events and exactly 1 **result** event will be triggered. After a **result** event, no further events will be generated for this geocast.
output	**collect**($id \in \mathrm{ID}_{request}$, $source \in$ Address, $m \in$ Message).
description	When receiving a geocast, receivers that are listening on the destination port have the option of sending back data. If they send data, the sender needs a way to collect it. The **collect** event is triggered by the geocast service as the responses come in, until **result** is triggered at t_{result}. id refers to the identifier used to initiate the geocast to which the message is a response.

output	**result**($id \in \text{ID}_{request}$, $R_{interested} \subset \text{Address}$, $R_{uninterested} \subset \text{Address}$, $V \in \text{MembershipView}$) at t_{result}.
description	The **result** event is generated at t_{result}. $R_{interested} \cup R_{uninterested}$ contains the addresses of confirmed receivers of the message and a set of membership tuples V (defined below). There may be a slight delay between the results being made available, and the application being able to process them. The application is responsible for anticipating and handling this delay. *id* refers to the identifier used to initiate the geocast of which this event is the result.
type	Timeliness = $(t_{result} \in \text{Time}, t_{target} \in \text{Time}, t_{delivery} \in \text{Time} \cup \{\perp\})$.
description	Timeliness is a tuple containing the relevant timeliness information of the geocast as discussed in Section 3.4.2. The following constraint must hold: $t_{delivery} < t_{result} \leq t_{target}$.
parameter	$R_{interested} \subset \text{Address}$.
description	If an application on vehicle i is listening on the destination port p when it receives a message generated by a **geocast** event at vehicle j, it has the option of sending a response. If it does, and the response arrives at V successfully, its address will be in $R_{interested}$ in the result to indicate that it was a participating receiver that successfully received and responded to the message. If a receiver elects not to respond, it will not appear in $R_{interested}$ despite having received the message.

parameter	$R_{uninterested} \subset$ Address
description	When the service receives a geocast message, but no application is listening on the destination port p of the message, it automatically sends an acknowledgement flagged as being from an uninterested vehicle. The acknowledgement means that the message was successfully received, but the vehicle was not interested. This allows the sender to distinguish between vehicles that were potentially interested, but failed to receive the message or respond, and vehicles that were not interested.
type	Membership = $(M \subset \text{Address}, A \in \text{Area}, t \in \text{Time})$. Instances written as: $\mathcal{M}(M, A, t)$.
description	A membership tuple is a reliable piece of information which specifies that no other vehicles than those whose identifiers are in M are in area A at time t. The *Area* type is defined in Appendix C.
type	MembershipView $\subset$ Membership.
description	A membership view is a set of membership tuples.

3.4.5 Receiver API

On the receiving end, the application must bind to and listen on a port, after which it can receive messages sent to that port. Since it can be useful to include data in the response to the geocast or change behaviour before responding, responses are not sent automatically. A separate **respond** primitive allows the application to manually respond. We use the notation defined in Appendix A to specify the API below.

Table 3.3: Receiver API specifications

input	**bind**(
	$bid \in \text{ID}_{bind}$,

$p \in$ Port).

description The application must bind to a port to receive messages sent to that port. It specifies a locally unique identifier and a locally unique port number. This is the same port used in the geocast by the sender and must be known in advance. There should also be an option to unbind, but this is not defined explicitly.

---

output **receive**(

$bid \in \text{ID}_{bind}$,

$rid \in \text{ID}_{response}$,

$m \in$ Message) at $t_{delivery}$ - if specified.

description If an application is bound to port p before the service receives a message destined for p, the message is delivered to the application through a **receive** event. The *bid* parameter refers to the identifier specified in the **bind** event.

The service generates *rid*, which the application can use for **respond** events. The implementation may also make the source address, target area, and timeliness specification available to the receiver, but these are not strictly necessary.

---

input **respond**(

$rid \in \text{ID}_{response}$,

$m \in$ Message).

description A receiving application can generate a **respond** event some time after a **receive**. To associate the response with a geocast, the application needs to specify the *rid* parameter that was given by the **receive** event.

The response serves to let the sender know that the message has been successfully received and processed by the receiving application and may include additional data. It is up to the application to ensure that it gives a response in time for it to be returned to the sender. If it does not use **respond** the sender will not know whether its geocast arrived.

3.4.6 Discussion

The group communication interface effectively fulfills the requirement for confirming the safety of actions in cooperative automated driving as defined in Section 3.2.5. It offers the ability to send a message to all the vehicles in an area at a particular time, offers vehicles the ability to respond, and can reliably confirm whether all eligible vehicles have done so. The remainder of this thesis will show how the model can be implemented using ordinary sensing and communication equipment, and how it can be applied to cooperative automated driving problems.

Chapter 4

Design and Implementation

This chapter presents a design and implementation of the Vertigo model to demonstrate the feasibility of implementing the guarantees of the model using a basic set of capabilities. The overall architecture of the implementation is shown in Figure 4.1, in which an arrow indicates a *uses* relationship. The implementation consists of three components: a beaconing service, a membership service, and a group communication service. Each component uses a set of capabilities offered by lower layers. The capabilities assumed to be present on the vehicle are ad-hoc networking, a set of LIDAR sensors, a position sensor, an orientation sensor, an accurate clock (not shown), and a road map. The implementation offers the sender and receivers APIs defined in Sections 3.4.4 and 3.4.5 to applications. The remainder of the chapter will describe the capabilities, with particular attention to the way the road map is defined, and the algorithms used by each of the components.

The actual code for the implementation is written in Java and is based on a generic capabilities interface that follows these definitions. The Java Topology Suite [Vivid Solutions Inc. 13] is used to implement several geometric algorithms. The implementation has been tested and evaluated on a cooperative automated driving simulator, which offers a simulated implementation of the capabilities and is described in Chapter 6. The

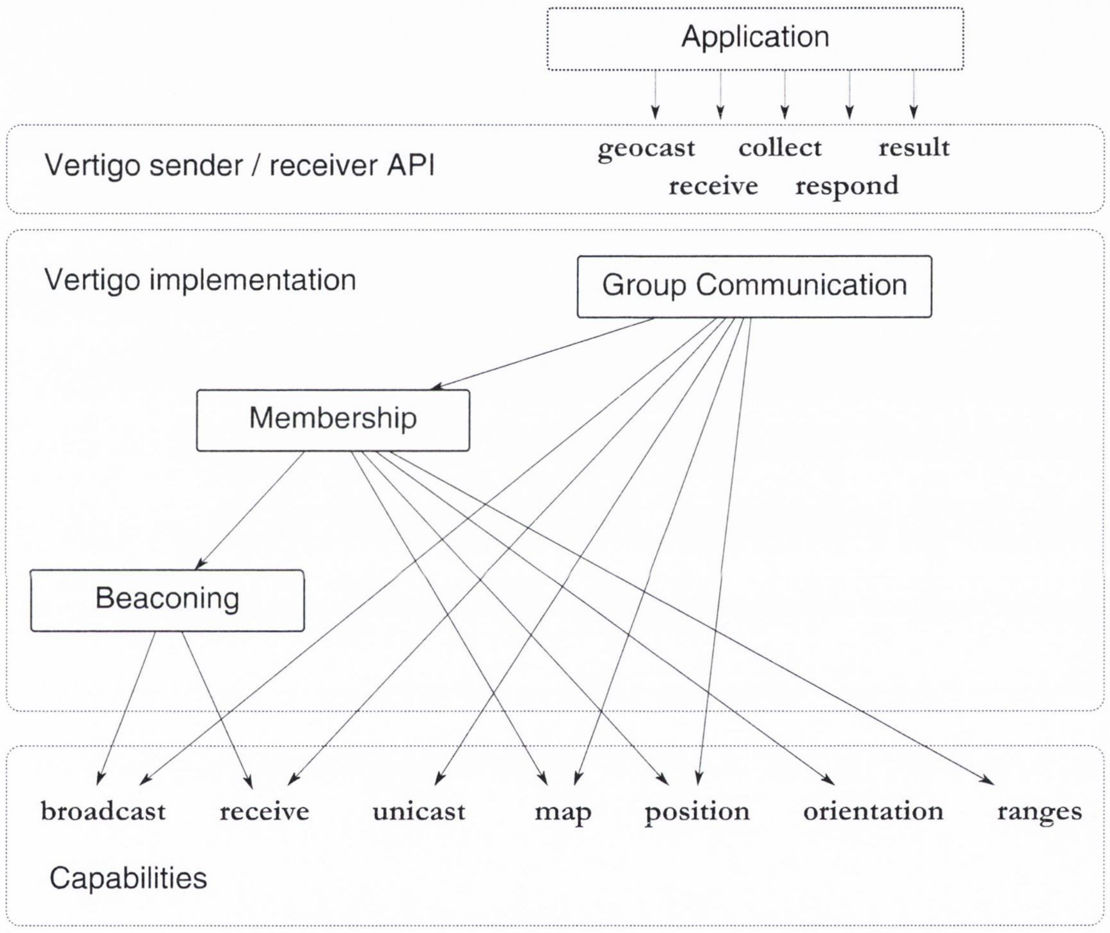


Fig. 4.1: Architecture of the Vertigo implementation

code for the implementation and the simulator can be found on:

`http://thesis.marcoslot.net/`.

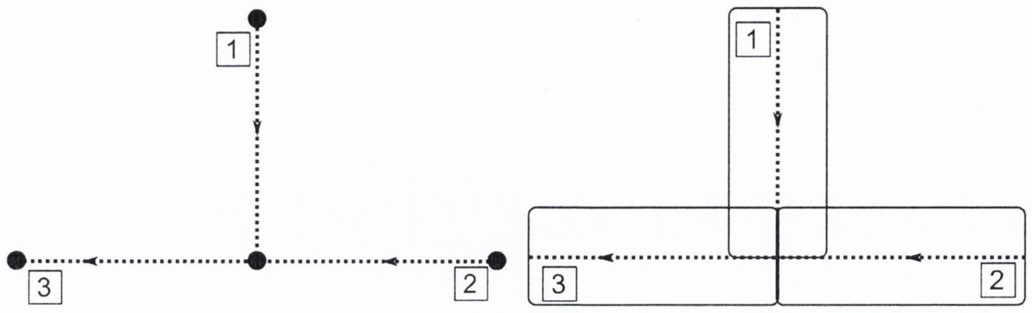
4.1 Capabilities

The capabilities of a vehicle can be seen as the environment in which the implementation operates. The implementation is effectively a translation of the services offered by lower layers to the service offered by the Vertigo communication model. Any vehicle which offers these capabilities can use the Vertigo implementation from this chapter. The assumed capabilities are a subset of the current capabilities of research vehicles [Levinson 11,Lidstrom 12]. To save space in this chapter, Appendixes B and C formally define the APIs of the capabilities shown in Figure 4.1.

An important aspect of the definition of these capabilities is that worst-case bounds on the accuracy of sensors are assumed. This assumption must hold to ensure the reliability of the implementation. However, as Chapter 6 shows, a relatively high inaccuracy can be tolerated without heavily impacting the ability of Vertigo to confirm success. Clock inaccuracy is not currently considered, but could be considered using the TrueTime API [Corbett 12] instead of absolute time stamps.

4.2 Road Map Area Definition

The way areas are represented is an essential aspect of the implementation. In a Cartesian coordinate system, areas can be represented as polygons or other two-dimensional shapes. However, in the context of vehicles driving on roads, such a representation would be impractical and inefficient. Only parts of an area that lie on a road are relevant, and the shape of the road itself almost never changes. Moreover, vehicles are heavily constrained in the way they move over roads, staying close to a specific path in a specific direction. These constraints can be exploited in order to define an area that is relative to a road map in a practical, and efficient manner. A specific point or section of a road map



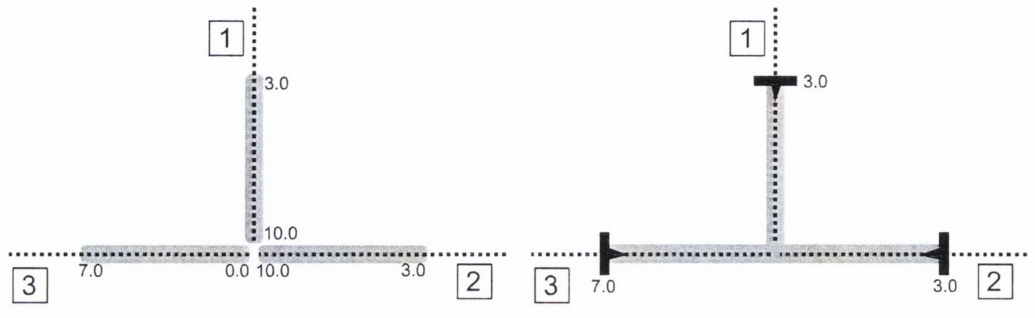
(a) Connectors and road segment paths (b) Road segment surfaces and paths

Fig. 4.2: Road map representation used by the Vertigo implementation consisting of road segments and connectors

can be defined in one dimension as an offset along the path of a road. Movement on a road can be reasoned about as a change in offset on the path. Where roads interconnect, their paths do as well, and movement can be represented as transitioning from one path to another.

Appendix C specifies the road map and different area representations. The road map is defined as a graph of road segments (edges) and connectors (nodes). Vehicles can drive from one road segment to another if there exists a connector between them (see Figure 4.2(a)). Road segments have a path, which is a two-dimensinal line string representing the shape of the road, and a surface area, which is a two-dimensinoal polygon (see Figure 4.2(b)). The paths and surfaces of connected road segments also connect, such that they form a contiguous geometric structure. Road segments can overlap to represent bridges and tunnels, as long as they are not connected by a connector. Road segments also define a maximum speed that vehicles on the road segment must adhere to.

There are multiple requirements on the specification of areas for the implementation of the Vertigo model. A mapping from sensor data to an area needs to be possible. Areas need to have a compact representation, such that they can be serialized



(a) Range-based area (b) Boundary-based area

Fig. 4.3: Area representations used in the Vertigo implementations

and communicated as part of messages. Areas also need to support spatial operations such as merging, and containment. To meet all these requirements, the implementation uses several different representations of areas, and defines translations between them. Some representations are only used as intermediate states and will be described as part of the algorithms. The primary representations used to represent areas in interfaces and messages are a one-dimensional range-based representation, and a one-dimensional boundary-based representation.

The range-based representation of an area consists of a set of (road segment identifier, start offset, end offset) tuples defining ranges of offsets which are included in the area on the network of paths. Offset is a distance over the path of the road segment from the start. Figure 4.3(a) shows an example of set $\{(1, 3.0, 10.0), (2, 3.0, 10.0), (3, 0.0, 7.0)\}$. The range-based representation allows for efficient geometric operations such as checking containment, and merging. The boundary-based representation consists of a set of (road segment identifier, offset, forward/backward) tuples defining the boundaries of a one-dimensional area on the network of paths. Forward/backward indicates which direction from the offset over the path is included in the area. Figure 4.3(b) gives an example of area $\{(1, 3.0, front), (2, 3.0, front), (3, 7.0, back)\}$. The boundary-based representation of an area can be compactly represented in messages, since large, complex areas can be captured by a small number of boundaries. Conversions between the representations are defined in the Appendix C and used in several places in the implementation.

4.3 Beaconing service

To participate in Vertigo, vehicles periodically broadcast their identifier, position measurement, and the time at which the position measurement was taken. Incoming beacons obtained using the **receive** interface are kept in a neighbourhood view, which can provide the last-known position of a vehicle, given its identifier. Before a beacon is sent using **broadcast**, additional data can be added to it by individual components, such as the membership service. The beaconing rate is configurable per application.

4.4 Membership Service

The membership service maintains a membership view, which is a set of membership tuples. The service generates local membership tuples from its sensor data, which it communicates to other vehicles by attaching them to beacons. Tuples from received beacons are stored in the membership view. When requested, the membership view is collapsed into a single tuple to allow it to be used for reliable success confirmation of a geocast query.

4.4.1 Sensing a Local Membership Area

When a local membership tuple is requested in order to transmit a beacon, or provide feedback on communication, a series of steps is performed to generate the membership tuple. The method used by the implementation is to convert LIDAR measurements into an area that is empty except for the vehicle itself, which gives a valid membership tuple for the current time. A series of steps is performed to convert the LIDAR measurements into an area of the one-dimensional range-based form, which can be compared to the

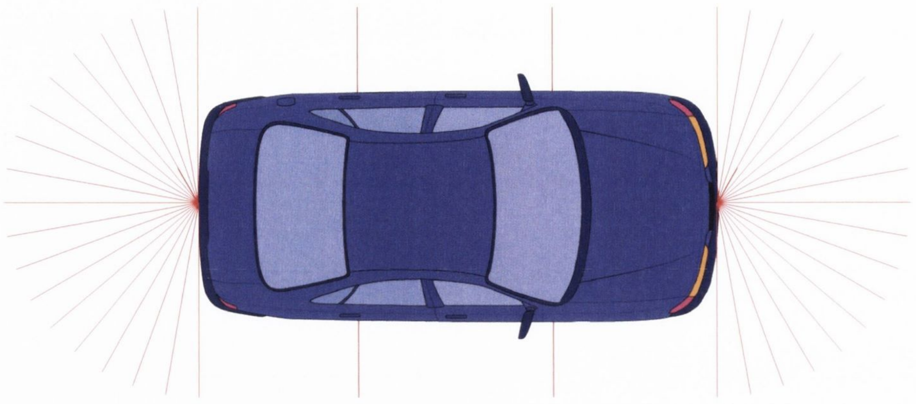


Fig. 4.4: LIDAR Sensors on the vehicle

target area of a geocast.

The algorithm performs the following steps:

1. Convert LIDAR beams to a polygon in a global coordinate frame
2. Subtract position and LIDAR uncertainty through polygon offsetting
3. Convert polygon to 2D road map areas
4. Convert 2D road map areas to a 1D range-based representation

The remainder of this section discusses each step in detail.

4.4.1.1 Convert LIDAR Beams to a Polygon

A vehicle is assumed to have a set of LIDAR sensors positioned on its outer-rim. The properties of the sensors, such as its relative position and orientation on the vehicle, the angular spacing of beams, and the range, are known by the implementation. The specific configuration used in current simulations is displayed in Figure 4.4. The configuration has 180° LIDARs on the back and front of the vehicle, which are modelled after the SICK laser range finders used in the DARPA grand challenge in 2005 [Buehler 07]. The

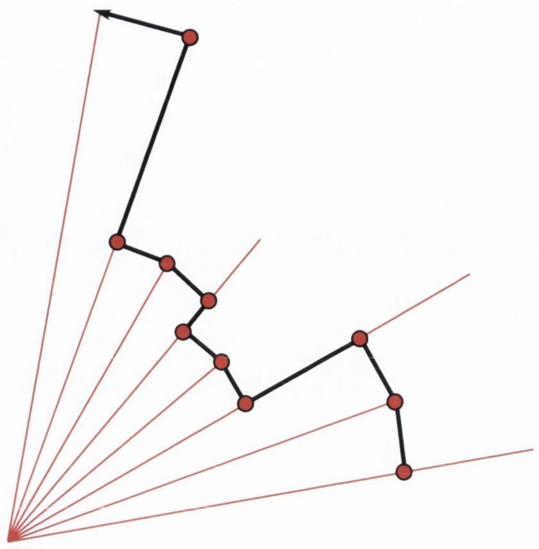


Fig. 4.5: Converting LIDAR beams to a ring of points

beams have an angular resolution of 1° and maximum range of 30m. Additionally, there are two single-beam LIDARs on each side of the vehicle to observe the area right next to the vehicle.

The set of beam measurements are obtained through the **ranges** interface. To convert the beam measurements into a two-dimensional area, they are traversed in circular fashion to form a ring of points. For every pair of adjacent beams, the projection of the tip of the shorter beam on the longer beam and the tip of the shorter beam are added to the ring. An example of the formation of such a ring is shown in Figure 4.5. The process is repeated until a ring is obtained that forms the outline of the polygon. An example of a complete polygon (yellow) extracted from LIDAR data (green) is shown in Figure 4.6[1]. The process is based on the assumption that the area between the beams is empty. To some extent this is a topological problem. Very small or pointy vehicles could fit in between a very narrow pair of beams, and different rotations of vehicles could be considered. The projection approach gives an approximation of the area that

[1]Map data ©2013 Google

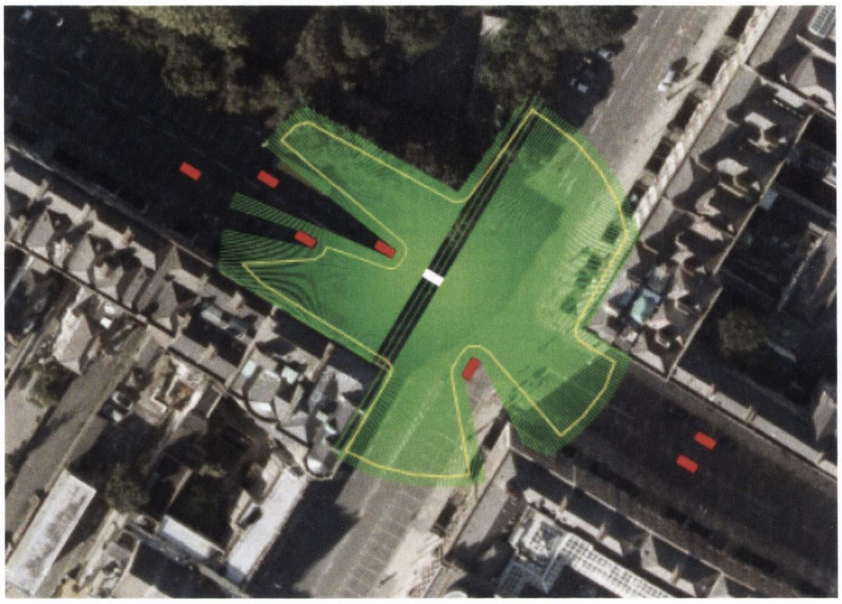


Fig. 4.6: Conversion of LIDAR data (green) to polygon (yellow)

is guaranteed to be empty given the length of the beams and rectangular vehicles, which can be made reliable by correcting the polygon for inaccuracy.

When all pairs of consecutive beams are processed the points form a polygon relative to the vehicle. The polygon is transformed to a global coordinate frame using the measured **position**, and rotated to the current **orientation** of the vehicle.

4.4.1.2 Accounting for Uncertainty

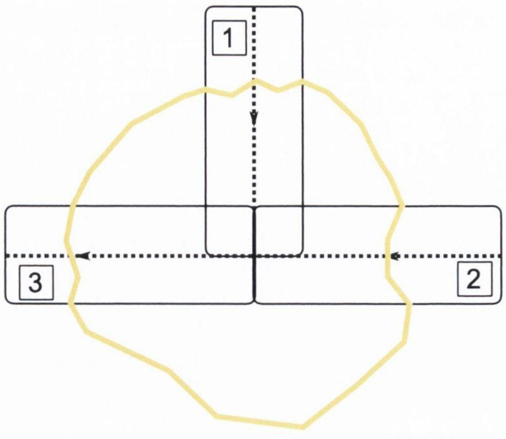
In the transformation of the polygon to the global coordinate frame, the position uncertainty needs to be taken into account. After transformation, the polygon may include points that were not empty, because the actual position of the vehicle is further from those points than the measured position. To correct for this difference, the worst-case uncertainty of the positioning sensor is subtracted from the polygon through a process called inwards polygon offsetting [Kim 98]. Offsetting moves any point on the boundary of the polygon inwards by a distance δ. Defined in another way, the obtained polygon is the intersection of all polygons that could be obtained by a transformation of a distance of at most δ. This means that the the polygon resulting from an inwards offset by δ meters is the area that is known to be empty even if the position of the original polygon lies up to δ meters away from the measured position.

In clear weather and within the maximum range, LIDAR measurements are very accurate, within a few millimetres of the actual value. LIDARs are less accurate in poor visibility, but in that case the measured values become lower than the actual distance to solid objects, which is not a problem from a safety perspective. A source of uncertainty that does pose a risk of overestimates is the time it takes to measure multiple LIDAR beams. The 180° laser range finders that were used in the DARPA grand challenge have a frequency of 75Hz, meaning vehicles could have moved $v_{max}/75$ since the oldest beam measurement (e.g. 53cm for v_{max} = 40m/s - highway speed). An additional 1m is taken away from the polygon to account for LIDAR uncertainty and other, smaller sources of uncertainty such as rounding errors and topological uncertainty.

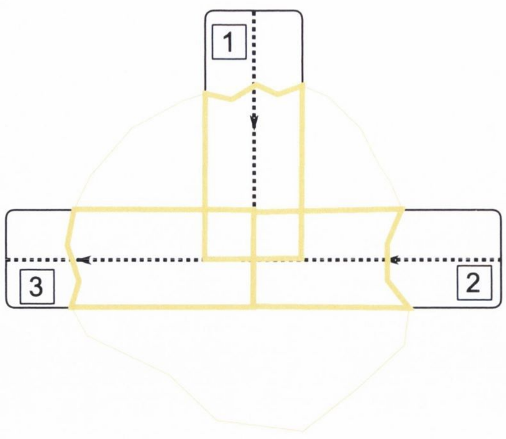
The offsetting is performed using a common geometry library. A conservative offsetting mode is used that preserves inwards pointing angles and avoids creating round curves with many points. Offsetting is the most computationally expensive step, but key to dealing with (bounded) inaccuracy.

4.4.1.3 Intersect with Road Map

The polygon obtained in the previous step gives an area in a two-dimensional Cartesian coordinate system. To be able to reason about this area from the perspective of a vehicle, it needs to be converted to an area that is relative to the road map. As previously mentioned, every road segment has a two-dimensional surface polygon associated with it. The outer-ring of the polygon is intersected with the surface of the current road segment and any connected road segments. Unconnected road segments are not considered since they may overlap in 3 dimensions, in which case the vehicle would falsely conclude that a road above or below it is empty based on its two-dimensional overlap with the polygon.



(a) Polygon overlaid on road map

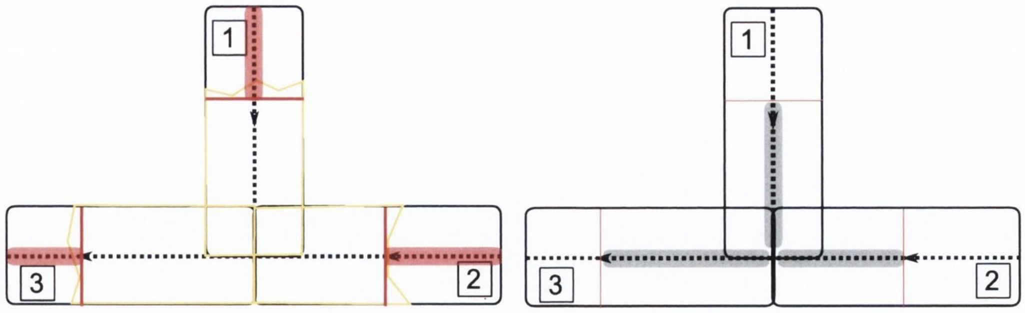


(b) Intersection with road segment surfaces

Fig. 4.7: Intersection of a measured polygon with the road map

The result of the intersection of the road segment surfaces with the empty area, is a set of polygons whose boundaries partially overlap with the boundaries of the road segment surfaces as shown in the example in Figure 4.7.

The polygons are converted into one-dimensional ranges by finding ranges for which the full width of the road segment surface is covered by the polygon. To do so, the polygon is separated into a set of line strings containing the points on the polygon that lie within the surface of the road segment. Any point on a line string that is not also a point in the road segment surface implies that the full width of the road is not covered. For each point in the line string, the projection onto the path of the road segment is computed and the corresponding offset from the start of the path. The minimum and maximum offset of the projections form a range r which is to be excluded from the final area. An example is shown in Figure 4.8(a), in which excluded ranges are marked by red lines. If two ranges r_1 and r_2 overlap they are merged by taking $(\mathbf{min}(r_1.start, r_2.start), \mathbf{max}(r_1.end, r_2.end), r_1.segment)$. The ranges are then sorted by offset, giving a lowest range r_{min} and the highest range r_{max}. The start and end of



(a) Ranges for which full width is not covered (b) Final range-based area after inversion

Fig. 4.8: Conversion of 2D road map area to 1D

the path are corner cases. If the start of the path is not inside a polygon, then r_{min} is expanded to $(0, r_{min}.end, r_{min}.segment)$. Likewise, if the end of the path is not inside a polygon, then r_{max} is expanded to $(r_{max}.start, l_{path}, r_{max}.segment)$ where l_{path} is the length of the path. Finally, the ranges are inverted; gaps become ranges and ranges become gaps to obtain a set of ranges for which the full width of the road segment is included in the measured polygon as per the example in Figure 4.8(b).

A concrete example of the conversion of a polygon to one-dimensional ranges is given in Figure 4.9, which shows a visualization of a running version of this algorithm. Vehicles are shown as red rectangles, the polygon derived from LIDAR measurements is shown in yellow, and the parts of paths that are covered by ranges in the resulting range-based area are marked in green.

The ranges for all road segments are combined to form an area A, which is added as part of a new tuple $\mathcal{M}(\{i\}, A, t)$, where i is the unique identifier of the vehicle, and t is the time at which the LIDAR measurements were taken.

The parts of the area where the full width of the road is not covered are lost in the conversion from 2D to 1D. This is acceptable from the perspective of membership, since it does not add any potentially unknown vehicles to the area. However, if two vehicles are driving next to each other on the same road segment they could never measure an empty

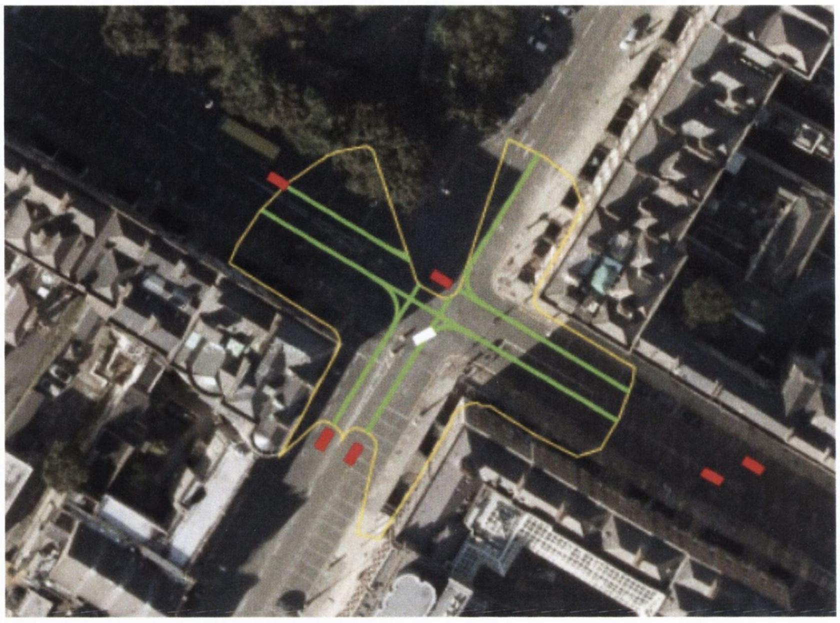


Fig. 4.9: Conversion of polygon to one-dimensional ranges

polygon that covers the full width of the road. The current approach assumes a fine-grained road map in which road segments describe individual lanes and no two vehicles could ever drive next to each other on the same road segment. If a road segment can contain multiple lanes, a preferable solution is to send the LIDAR data in compressed form as part of beacons, compute the polygons from the received data, compute the union of the polygons (considering the effects of delay between measurements through offsetting), and then perform the conversion described in this section. A demonstration of this approach is given in [Slot 11b]. The drawback is that sending LIDAR data generates far more network traffic, and merging polygons adds a computationally expensive step to the process, so expensive that real-time simulations are no longer feasible. However, the solution may be preferable in a setting with coarse-grained, or inaccurate maps. This chapter describes the implementation as it used in the evaluation in Chapter 6, which uses the more efficient approach to allow real-time simulations.

4.4.2 Beaconing Membership Tuples

A vehicle i which establishes that area A is empty at time t can generate the tuple $\mathcal{M}(\{i\}, A, t)$. This tuple can be attached to beacons in order to share it with neighbours. To represent the area compactly, the boundary-based representation is used. The boundaries are represented as (road segment id, offset, forward/backward) tuples, which are serialized using 4 bytes for the road segment identifier, 4 for the offset, and 1 for the direction, to a total of 9 bytes per boundary. The total network overhead of sending a membership tuple depends mainly on the number of boundaries, and is studied further in Chapter 6.

Vehicles synchronize the time at which they generate the membership tuple to send in the beacon using their clocks. The benefit of this is that the time difference between tuples can be kept at a minimum, which avoids big gaps resulting from decay.

Neighbours of vehicle i that receive the beacon containing the membership tuple store it in their local membership view. Periodically, the membership view is cleaned by removing all beacons whose area is empty when decayed to the current time, since they no longer convey any information.

4.4.3 Merging 1D Road Map Areas

The tuples that are received from other vehicles can be merged into a single tuple $\mathcal{M}(M, A, t)$ when needed to confirm success for a **geocast**, as defined in Section 3.4.4. For this purpose, the range-based form of A is used. The merging operation is a logical union between an area $A_{merged} = A_1 \cup A_2$ such that any point in A_{merged} is in either A_1 or A_2 and vice versa.

Merges are performed using the range-based area representation, which defines an area as a set of ranges. A new range r can be added to an area by joining it with any existing ranges it intersects with before adding it to the set. To merge two areas, all the ranges in one area are added to the other according to the algorithm below.

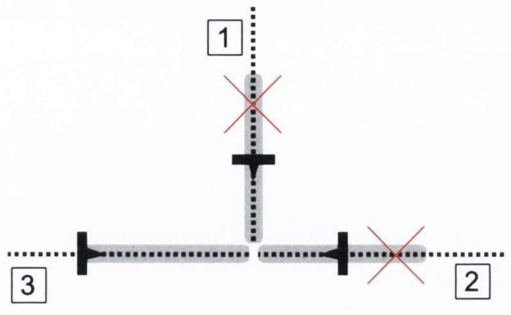
```
func merge($A_1 \in$ RangeArea1D, $A_2 \in$ RangeArea1D).
$A_{merged} \leftarrow A_1$.
for all $r \in A_2$ do
    $o_{min} \leftarrow r.start$.
    $o_{max} \leftarrow r.end$.
    for all $l \in A_{merged}$ where $l.segment = r.segment$ do
        if intersects$(r, l)$ then
            $o_{min} \leftarrow$ min$(o_{min}, l.start)$.
            $o_{max} \leftarrow$ max$(o_{max}, l.end)$.
            $A_{merged} \leftarrow A_{merged} \setminus \{l\}$.
        end if
    end for
    $A_{merged} \leftarrow A_{merged} \cup \{(o_{min}, o_{max}, r.segment)\}$.
end for
return $A_{merged}$.
end merge.
```

```
func intersects($r \in$ RoadSegmentRange1D, $l \in$ RoadSegmentRange1D).
return $(r.segment = l.segment) \wedge (r.end \geq l.start) \wedge (r.start \leq l.end)$.
end intersects.
```

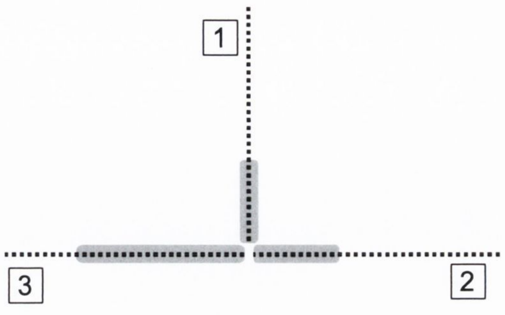
All ranges from A_2 are added to all ranges from A_1 to form A_{merged}. The algorithm from Section 3.3.4 to merge multiple membership tuples can then be used.

4.4.4 Decaying 1D Road Map Areas

To merge membership tuples or shift them forward to the target time t_{target} of the **geocast** before being passed to the application in the **result** event, a decay function needs to be defined. To decay a membership tuple $\mathcal{M}(M, A, t)$, δ is added to t and A is



(a) Boundaries moved inwards in decay



(b) Range-based area after decay

Fig. 4.10: An example of decay of a one-dimensional road map area

reduced in size to account for vehicles that may have been driving at maximum speed from the boundaries of A between time t and time $t + \delta$, reducing the size of the area which is known to be empty except for vehicles whose identifier is in M.

The algorithm for shrinking an area first obtains the boundaries of the area using the **rangesToBoundaries** function from Appendix C and then moves boundaries by a distance of at most $v_{max} \times \delta$ in the direction of the road segment (which may be bi-directional) as shown in the example in Figure 4.10. The distance moved is removed from the area. If a connector is encountered, the remaining δ is calculated and a new walk is started from the end of each connecting road segment by inserting a new boundary. The algorithm can also expand the area if δ is negative, in which case the distance walked in the opposite direction of the road segment is added to the area. The full algorithm is given below.

```
func shift(
    $area \in$ RangeArea1D,
    $\delta \in \mathbb{R}^+$).
$boundaries \leftarrow$ rangesToBoundaries($area$).
for all $boundary \in boundaries$ do
```

```
  **shiftFromBoundary**$(area, boundary, \delta)$.
**end for**
**end shift**.

**func shiftFromBoundary**(
  $area \in$ RangeArea1D,
  $boundary \in$ Boundary1D,
  $\delta \in \mathbb{R}^+$).
$segment \leftarrow segments[boundary.segment\_id]$
$distance \leftarrow |\delta| \times segment.v_{max}$.
**if** $\delta > 0$ **then**
  $forwardDecay \leftarrow boundary.inclusive = front$.
**else**
  $forwardDecay \leftarrow boundary.inclusive = back$.
**end if**
**if** $((segment.direction = backward \vee segment.direction = both) \wedge \neg forwardDecay) \vee$
$((segment.direction = forward \vee segment.direction = both) \wedge forwardDecay)$ **then**
  $remainingDistance \leftarrow distance$.
  **if** $forwardDecay$ **then**
    {Offset of boundary increases}
    $start \leftarrow boundary.offset$.
    **if** $start + distance \leq$ **length**$(segment.path)$ **then**
      {Shifting ends on this segment}
      $end \leftarrow start + distance$.
      $remainingDistance \leftarrow 0$.
    **else**
      {Shifting continues onto other segments}
      $end \leftarrow$ **length**$(segment.path)$.
    $remainingDistance \leftarrow distance - (end - start)$.
  **end if**
**else**
  {Offset of boundary decreases}
  $end \leftarrow boundary.offset$.
  **if** $end - distance \geq 0$ **then**
    {Shifting ends on this segment}
    $start \leftarrow end - distance$.
    $remainingDistance \leftarrow 0$.
  else
    {Shifting continues onto other segments}
    $start \leftarrow 0$.
    $remainingDistance \leftarrow distance - (end - start)$
  **end if**
**end if**
**if** $\delta > 0$ **then**
  $area \leftarrow area \setminus \{(segment\_id, start, end)\}$. {Shrink the area}
  $\delta \leftarrow \delta - (end - start)/segment.v_{max}$. {Compute remaining $\delta$}
**else**
  $area \leftarrow area \cup \{(segment\_id, start, end)\}$. {Expand the area}
  $\delta \leftarrow \delta + (end - start)/segment.v_{max}$. {Compute remaining (negative) $\delta$}
**end if**
**if** $remainingDistance > 0$ **then**
  **if** $forwardGrowth$ **then**
    **shiftFromConnector**($area, segment.to, \delta$).
  **else**
    **shiftFromConnector**($area, segment.from, \delta$).
  **end if**
```

```
    **end if**
  **end if**
  **end shiftFromBoundary**.

  **func shiftFromConnector**(
    $area \in$ RangeArea1D,
    $connector\_id \in \mathrm{ID}_{connectors}$,
    $\delta \in \mathbb{R}^{+}$).
  $connector \leftarrow connectors[connector\_id]$.
  **for all** $segment\_id \in connector.segments$ **do**
    $segment \leftarrow segments[segment\_id]$.
    **if** $connector\_id = segment.from$ **then**
      **if** $\delta > 0$ **then**
        {Shrink area further from start of segment}
        **shiftFromBoundary**($area, (segment\_id, 0, front), \delta$).
      **else**
        {Grow area further from start of segment}
        **shiftFromBoundary**($area, (segment\_id, 0, back), \delta$).
      **end if**
    **else**
      **if** $\delta > 0$ **then**
        {Shrink area further from end of segment}
        **shiftFromBoundary**($area, (segment\_id,$ **length**$(segment.path), back), \delta$).
      **else**
        {Grow area further from end of segment}
        **shiftFromBoundary**($area, (segment\_id,$ **length**$(segment.path), front), \delta$).
      **end if**
    **end if**
```

```
end for
end shiftFromConnector.
```

The outcome of the shift algorithm is used to construct a decayed tuple as described in Section 3.3.3.

4.4.5 Collapsing Membership Tuples

When the **result** event occurs (see Section 3.4.4 for the definition), the set of receivers for a geocast needs to be compared to the membership view. At that point, the tuples are merged according to the algorithm in Section 3.3.4, which orders the tuples chronologically and repetitively decays the oldest to the second oldest tuple to merge them, using the decay and merge algorithms for road map areas. Finally, the resulting tuple is decayed to the target time t_{target}.

The membership tuple is requested by the group communication service to confirm a set of receivers R is a superset of all the vehicles in a given target area A_{target} at a given target time t_{target}. Only tuples for which the set of members M is a subset of the set of receivers R are considered when merging tuples. If membership tuples for which M is not a subset of R are considered in the merge, then $R \supseteq M$ would *not* hold for the merged tuple either, which means that the success of a query cannot be confirmed. The approach of filtering by R is optimal in the sense that it always confirms success if possible. By applying the filter, success becomes detectable solely by checking whether, given a tuple $\mathcal{M}(M, A, t_{target})$, the membership area A contains a target area A_{target}. If this is the case, any vehicles in A_{target} at t_{target} must be in R. If a vehicle j is in the area but not in R, then either there is no membership tuple containing j and the vehicle would create a gap in the LIDAR observations of other vehicles and the resulting A, or there is such a tuple, but it is not considered because $j \in M$ and $j \notin R$, therefore $R \not\supseteq M$.

4.5 Group Communication Service

The group communication service is the component that implements the Vertigo API (see Sections 3.4.4 and 3.4.5). It offers the **geocast**, **collect**, and **result** interfaces for senders, and the **receive** and **respond** interfaces for receivers.

4.5.1 Starting a geocast query

A **geocast** event at time $t_{geocast}$ triggers a query to vehicles in a delivery area, which is derived from the target area defined by the applications using the target time. The target area A_{target} in the **geocast** interface uses the one-dimensional range-based format. The aim of the geocast is to reach all vehicles that will be in the target area A_{target} at time t_{target}. To ensure that all those vehicles can be reached, the boundaries of the target area are expanded by $v_{max} \times (t_{target} - t_{geocast})$. The expansion algorithm is effectively the inverse of the decay and implemented by the **shift** function from section 4.4.4, which is used with $\delta = t_{geocast} - t_{target}$ (which is negative).

The delivery area resulting from the algorithm is serialized using its boundary-based form, which is obtained using the **rangesToBoundaries** function from Appendix C. A query is constructed that includes the message from the application, the delivery area, a destination port, the unique identifier of the vehicle, a sequence number identifying the query, the result deadline, and the (optional) delivery time. The query is send to surrounding vehicles using the **broadcast** interface.

4.5.2 Receiving and responding to a geocast query

When a vehicle receives a query, it converts the delivery area back into its original range-based form using the breadth-first-search algorithm described in Appendix C. If the vehicle does not view itself as being inside the delivery area based on its latest sensor reading, it discards the message. It also discards the message if the result deadline or delivery time have passed. If the message has a delivery time, it is added to a priority queue to guarantee that the delivery order is consistent across receivers. Queries from the queue are passed to the application when the clock is equal to or greater than the delivery time. Queries whose delivery time lies before that of the last executed event are discarded. The order of messages is effectively guaranteed by exploiting the fact that omission failures are expected. This guarantee can be useful to applications, but is a much weaker guarantee than a totally ordered multicast. If no delivery time is specified, the message is passed on directly to the application listening on the destination port through the **receive** interface.

The application obtains the message using the **receive** interface and performs some arbitrary processing. It then uses the **respond** interface to pass a small or empty response back to Vertigo at time $t_{response}$. If $t_{response} > t_{result}$ the response is discarded, since it cannot be delivered in time for the result deadline. Otherwise, the response is sent back over **unicast** to the sender of the query. Many vehicles will receive the query and send a response at approximately the same time, potentially causing contention on the wireless network. To avoid contention as much as possible, the message is sent back with a random delay of at most $t_{result} - t_{response}$. The response message also contains the identity of the responder, the sequence number of the query, and a flag indicating that the application processed the message (was 'interested'). If no application is listening on the destination port, an empty response is sent back with a flag indicating that no application processed it ('not interested').

A query causes multiple vehicles to all send responses to a single source, which is known as an ACK implosion and creates a sudden burst of network traffic. This problem is primarily dealt with by keeping responses very small (16 bytes header and application-level response) and introducing a random delay (jitter) before responding. However, there is no reliable alternative to explicit acknowledgement to guarantee to the source that the message arrived. Negative ACKs are sometimes used to prevent ACK implosion in multicast [Sobeih 04], but the negative ACK might be lost, causing the sender to falsely conclude the transmissions was successful. Moreover, negative acknowledgements can only be applied if receivers already know that a transmission is about to happen, otherwise they would not know when to send the negative ACK.

4.5.3 Feedback on a geocast query

If the source of the query receives a response message, and the receiver was interested, it is passed to the application through the **collect** interface and the unique identifier of the receiver is added to $R_{interested}$, which is kept per query. If the receiver was not interested, its identifier is added to $R_{uninterested}$. At the result deadline, $R_{interested}$ and $R_{uninterested}$ are passed to the initiating application with the latest membership tuple (collapsed to $R = R_{interested} \cup R_{uninterested}$) through the **result** interface, which completes the query.

To confirm successful delivery of the query, it needs to hold true that given a membership tuple $\mathcal{M}(M, A, t)$ and a target area of the **geocast** A_{target}: $A \supseteq A_{target}$. In other words, the target area is contained by the membership area. Both areas are represented as in a set of ranges. A **contains** algorithm for range-based areas is given below.

```
func contains($A \in$ RangeArea1D, $A_{target} \in$ RangeArea1D).
for all $r_{target} \in A_{target}$ do
  $contained \leftarrow$ false.
  for all $r \in A$ where $r.segment = r_{target}.segment$ do
    if $(r.segment = r_{target}.segment) \wedge (r.start \leq r_{target}.start) \wedge (r.end \leq r_{target}.end)$
    then
      $contained \leftarrow$ true.
    end if
  end for
  if $\neg contained$ then
    return false.
  end if
end for
```

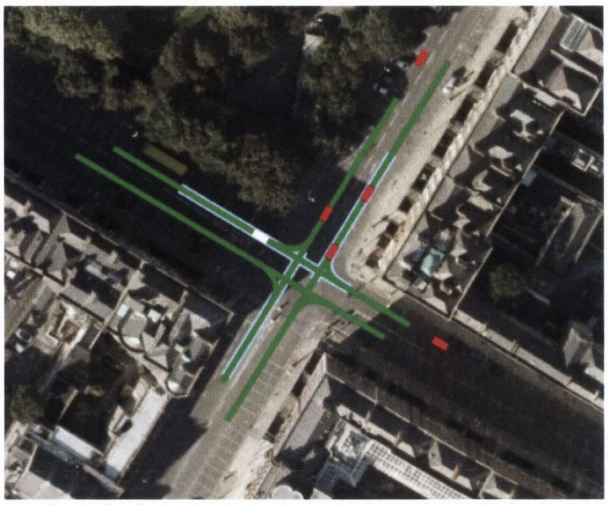


Fig. 4.11: A successful query, the membership area (green) covers the target area (blue)

```
return true.
end contains.
```

If indeed $R_{interested} \cup R_{uninterested} \supseteq M \wedge$ **contains**(A, A_{target}), then success of the query is reliably confirmed. Figure 4.11 shows an example of a successful query by the white vehicle. The membership area A is displayed in green and covers the target area A_{target} displayed in blue. In the example in Figure 4.12, the membership area A (now red) does not fully cover the target area, due to an omission failure.

4.5.4 Forwarding

If the **broadcast** and **unicast** operations offered by the network interface pass messages over a single hop, then they have an expected range of 100-300m [Demmel 12]. However, the delivery area may exceed this range. In that case, one way of reaching other vehicles in the delivery area is to forward the message over multiple hops. This is a challenging problem in a vehicular networks, in which both high contention and high partitioning are common. Several forwarding protocols have been developed that can adapt to both

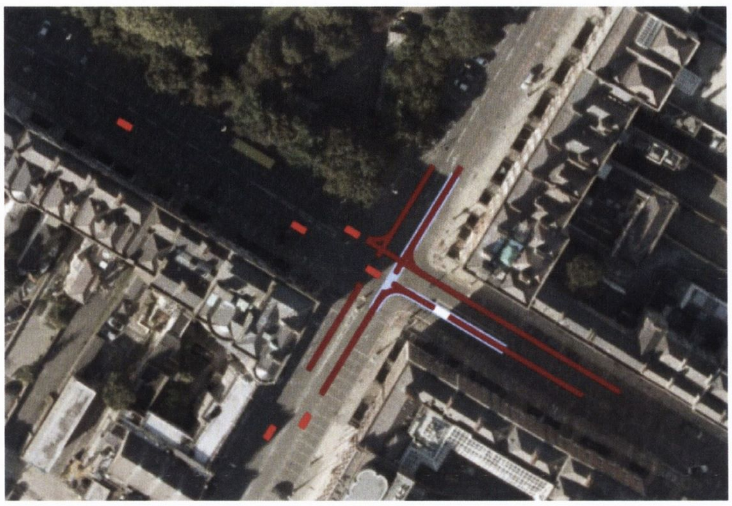


Fig. 4.12: Unsuccessful query due to omission failure, the membership area (red) does not cover the target area (blue)

conditions, most notably DV-CAST [Tonguz 10], which can switch between 'instant' forwarding and store-carry-forward, and AckPBSM [Ros 09], which combines forwarding based on the observed network topology with periodic acknowledgements to recover from partitions. For the purposes of Vertigo, DV-CAST is insufficient, since it only supports forwarding in the opposite direction of traffic. An implementation of AckPBSM was added to Vertigo in order to support query forwarding, but it is not used in any existing scenarios, since we have not found a meaningful use case for large target areas. This section briefly describes the AckPBSM protocol and adaptations made to make it suitable for Vertigo.

AckPBSM is a broadcasting protocol with two modes of forwarding. In the topology-based mode of forwarding, a vehicle i that receives a message starts a timer to forward the message with a low random delay if it deems itself part of the connected dominating set (CDS), an efficient broadcasting structure first proposed by [Cardei 02]. Receivers that are not in the CDS start a timer with a higher delay to fall back to if no vehicle in the CDS was reached. When a receiver sees a message being forwarded to its neighbours by another vehicle, it cancels its own timer. The second mode of forwarding supported by AckPBSM is based on acknowledgements of ongoing broadcasts that are added to beacons until the expiration time (t_{result} in case of Vertigo). If a vehicle i that received a message m receives a beacon from vehicle j that does not contain an acknowledgement for m, then vehicle i starts a timer with a random delay and broadcasts m if no other vehicle does. A vehicle that moves from a partition A to partition B will therefore quickly receive any messages that have only been forwarded within B and share messages that have only been forwarded in A. AckPBSM is meant for unbounded broadcast, but can be modified for geocast. When geocasting, a receiving vehicle i only starts a timer to forward a message if i is present in the delivery area, and only reacts to missing acknowledgements from another vehicle if that vehicle is in the delivery area for the message according to the position in its beacon.

In addition to forwarding the query, the responses from vehicles also need to be forwarded back to the source of the query. To this end, a protocol for end-to-end acknowledgement of geocast can be used [Slot 11a]. The protocol partially avoids the problem of ACK implosion by aggregating multiple acknowledgements before forwarding them. The protocol triggers forwarding of acknowledgements by simulating a timed wave that goes from the source to the boundaries of the delivery area and back. When the calculated position of the returning wave passes a vehicle, it forwards its acknowledgement and any acknowledgements it received from other vehicles towards the source. Acknowledgement are forwarded through (single hop) unicast, the destination of the unicast is a vehicle selected from neighbours that are closer to the source than the sender using consistent hashing to map the hash of (source identifier, sequence number) to the hash of the neighbour identifier. It is very likely that two vehicles will choose the same aggregator if their neighbourhood views overlap, which improves aggregation. The wave continues towards the original position of the source. When it arrives there, the source may have moved away. To deal with mobility, the source sends unacknowledged geocasts towards its original position for the duration of the query (until t_{result}). In the Vertigo implementation, acknowledgements can be small responses and membership tuples are included in messages. The tuples are aggregated by merging them at every hop.

The feasibility of using forwarding in a Vertigo implementation is shown by the forwarding speed of at least 300m/s found in [Slot 11a], which is far greater than the maximum speed of vehicles. Provided there are no network partitions, areas could be arbitrarily large and there would still be enough time to gather membership information that will only partly be decayed by the time it arrives at the source. Tuples need to be decayed due to the delay in forwarding, but merging tuples moves the boundaries of the membership area further apart. The fractional rate of decay is inversely proportional to the size of the membership area, which is proportional to the number of vehicles involved.

4.6 Discussion

Reliable success confirmation is the key aspect of the Vertigo communication model. Given that the implementation uses explicit acknowledgement, the set of receivers provided by the **result** event is reliable, since every vehicle from which a response was received must have also received the message. The reliability of the membership tuples relies on the correctness of the inaccuracy bounds of sensors and the correctness of the road map. The inaccuracy bound should be conservatively adapted to the ability of the vehicle to obtain accurate readings. For example, if a positioning system lacks the necessary satellite signals for accuracy, this can be reflected by using a high bound or inability to produce membership tuples. In this way, success confirmation will only be provided if it is reliable enough to be depended on for safety, even if it may lead to a reduced rate of success. The evaluation of the Vertigo implementation given in Chapter 6 will show the bounds on position inaccuracy and sensor range under which Vertigo can achieve a high rate of success.

Chapter 5

Application

The Vertigo model exists to help solve the problem of cooperative automated driving. To demonstrate its applicability to this problem, this chapter gives the design and implementation of a protocol that uses the Vertigo model to coordinate arbitrary intersection crossings in a safe, distributed manner. The protocol assumes vehicles follow *tracks*, which intersect in *conflict areas*, and uses Vertigo to request an *allocation* for entry into a conflict area from surrounding vehicles.

5.1 Tracks

The road network model that the application uses is based on tracks, interconnected line strings. The start and end points of a track lie on other tracks to form a network. Vehicles move through the road network by following a sequence of tracks, which may only be followed in one direction. Like road segments in the Vertigo implementation, tracks have a surface area, which is found by offsetting the line string by half the lane width in both directions to form a polygon. Unlike road segments, tracks do not form a graph, but tracks can be converted to road segments by splitting the track at the points where other tracks connect to it, and placing connectors between the segments. Areas on tracks are represented as a set of track ranges, which are (track id, start offset, end

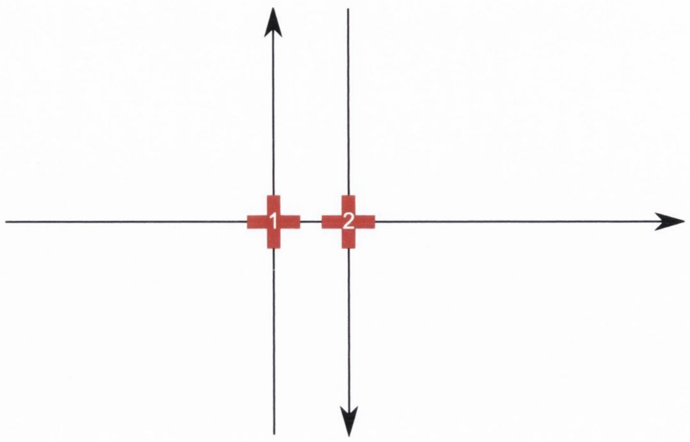


Fig. 5.1: Conflict areas at the intersection of tracks

offset) tuples specifying the part of the track contained in the area. Positions on tracks are represented as offsets from the start of a track combined with a track identifier.

5.2 Conflict Areas

Vehicles have a road map that contains the full set of tracks that can be followed, as well as a set of conflict areas. A conflict area is a pair of track ranges that is generated by intersecting the surface area of one track, with the line string of another. An example of conflict areas is shown in Figure 5.1. A conflict area represents a place where two tracks are less than a lane-width from each other. Vehicles cannot drive through a conflict area at the same time. A conflict area can therefore be seen as a resource for which mutual exclusion must hold at all times in order to ensure safety.

5.3 Empty Trajectory

The trajectory of a vehicle is represented as a sequence of tracks to follow from the current position of the vehicle. A subsequence of length f from the start of the trajectory is measured as empty using the LIDAR sensors of the vehicle, and is known as the empty

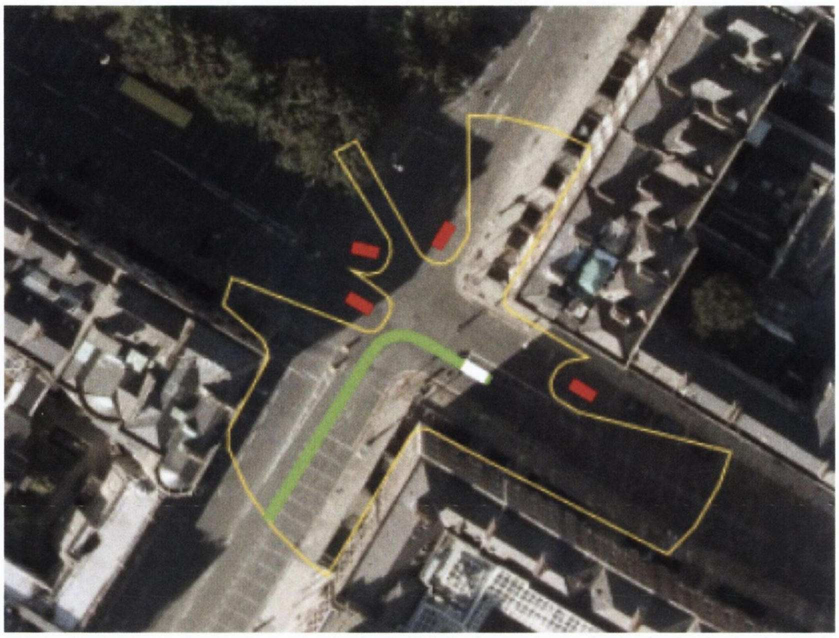


Fig. 5.2: Empty trajectory (green) provides forward distance

trajectory. The empty trajectory is computed using the one-dimensional empty area as described in Section 4.4.1. An example is shown in Figure 5.2[1], in which the yellow line represents the empty area, and the green line the empty trajectory. The value of f gives a higher-level notion of forward distance than a pure ranging sensor measurement, since it also consider turns. The empty trajectory is also used to detect upcoming conflict areas. If the empty trajectory intersects with a conflict area, f is reduced to the distance to the start of the conflict area, unless an allocation for the conflict area is obtained using the protocol described in Section 5.5.

5.4 Moving Forward

To decide on acceleration and deceleration, vehicles use the Intelligent Driver Model (IDM) [Treiber 13]. IDM is meant as a model for human driving behaviour in traffic

[1]Map data ©2013 Google

simulations, but can also be used to make acceleration decisions in automated driving. The main inputs into the model are the current speed and the distance to the vehicle ahead, which is obtained by computing the length f of the empty trajectory. The model also requires the speed of the vehicle ahead, which is derived from repeated distance measurements.

Under normal circumstances, the driver model is used to decide the acceleration, with two exceptions that are necessary to deal with stop-and-go in intersections. If the vehicle has sufficient forward distance, but is nearly stopped (speed is less than 5m/s), then it uses maximum acceleration to start. If forward distance is approaching the computed minimum stopping distance of the vehicle, it uses maximum deceleration. The latter is an important part of preserving safety, which is not guaranteed by IDM.

5.5 Coordination Protocol

When a vehicle approaches a conflict area, it uses a coordination protocol to attempt to obtain an *allocation* for a *conflict trajectory*, which is a specialized data structure for describing the intentions of the vehicle with regards to conflict areas. A vehicle i sends an *allocation request* using Vertigo, asking to enter the conflict trajectory in a specified time frame. Receivers of the request respond using an `accept`, `reject`, or `tentative` message. If all vehicles responded with either an `accept` or `tentative` message, it is safe for the vehicle to proceed enter the conflict areas described in the conflict trajectory, provided that vehicles that sent a `tentative` response have passed. The remainder of this section describes the details of the protocol.

5.5.1 Initiating the Coordination Protocol

While driving, a vehicle i periodically evaluates whether the empty trajectory intersects with any conflict areas. If a conflict area on the empty trajectory lies less than z meters away (where z is a configurable parameter), the vehicle initiates the coordination protocol. By starting the protocol only when a conflict area is in the empty trajectory, vehicle i is prevented from requesting an allocation for a conflict area before vehicles directly in front of i. By starting the protocol only when a conflict area lies less than z meters away, the set of vehicles competing for the same conflict area at the same time is bounded. A lower z reduces the number of vehicles with which to coordinate, and the probability of omission failures occurring. However, if z is too low, vehicles will always have to slow down or stop before being able to start the protocol to obtain an allocation for a conflict area, which increases their travel time. An appropriate value of z may depend on the scenario and requires tuning.

5.5.2 Conflict Trajectory

The first step of the coordination protocol is for vehicle i to generate a conflict trajectory T_i. The conflict trajectory is a sequence of (track range, set of conflict areas) tuples. The example in Figure 5.3 consists of 5 of these tuples, two of which contain a conflict area. Each tuple contains the exact set of conflict areas that intersect with the track range. Note that conflict areas may overlap, leading to different tuples for the parts where the conflict areas overlap and where they do not overlap. The sequence starts l_{commit} meters before the first conflict area in the empty trajectory, where l_{commit} is a configurable parameter and must be smaller than z. This area is known as the *commit area* and entering it is used to signal committing to a conflict trajectory. The sequence continues until $l_i + d_{min} + l_{commit}$ of conflict-free space is encountered, where l_i is the length of vehicle i and d_{min} the minimum distance vehicles need to keep at all times. This will provide the vehicle with enough space to stop before - and, if necessary, request a new allocation of - any subsequent conflict areas. Otherwise, vehicle i might be forced to stop on one conflict area waiting for another vehicle j on a second conflict area, while j might be waiting for i, creating a deadlock.

The conflict trajectory can be serialized efficiently for use in messages. Since the conflict areas are known globally, only a sequence of track ranges has to be serialized,

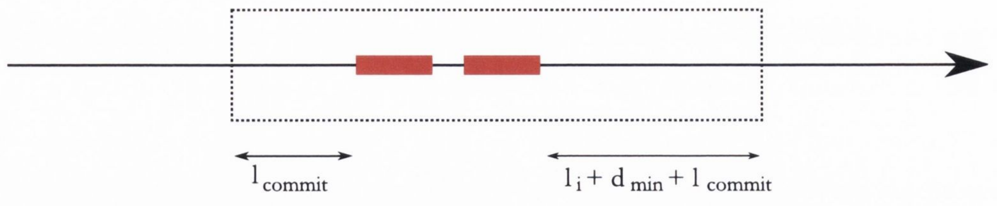


Fig. 5.3: A conflict trajectory through 2 conflict areas with a commit area of length l_{commit} and stopping distance $l_i + d_{min} + l_{commit}$

which consist of a track identifier and 2 offsets, which are serialized as 12 bytes in the implementation. Construction of a conflict trajectory from a sequence of track ranges is relatively efficient, since a mapping of track identifiers to tracks is available.

The conflict trajectory is effectively the implementation of an intended *action* (Q) as defined in the System Model chapter in Section 3.2.3.

5.5.3 Allocation State

To keep track of the state of an allocation, a vehicle keeps a state machine shown in Figure 5.4, which starts in the `initial` state. After an allocation request is sent using Vertigo, the state machine is in the `pending` state. At the result deadline of the Vertigo query, the state machine proceeds to the `obtained` state if the request was successful, or the `initial` state otherwise, meaning a new allocation request will have to be made. If the vehicle enters the first track range of the conflict trajectory in the allocated time frame, then the state machine proceeds to the `committed` state, or the `initial` state otherwise. Finally, once a vehicle moves past the last conflict area in the conflict trajectory it proceeds to the `released` state.

5.5.4 Sending an Allocation Request

The conflict trajectory T_i generated by vehicle i is serialized and added to an allocation request, which also includes a sequence number to identify the allocation, a start time

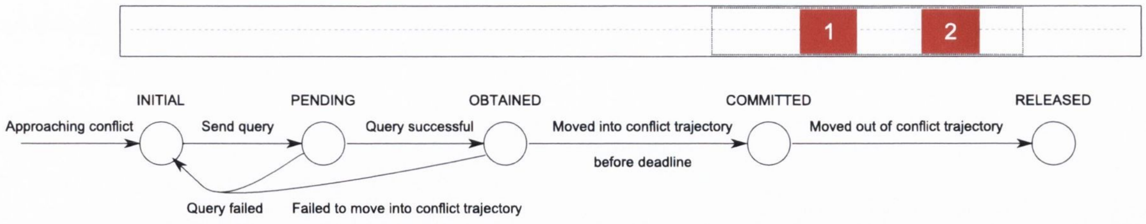


Fig. 5.4: The allocation state machine exemplified with a conflict trajectory

t_{start}, and an end time t_{end}. The meaning of an allocation request is that the vehicle is requesting to *enter* the first track range in the conflict trajectory (which is in the *commit area*) after t_{start} and before t_{end}. If other vehicles approve and the vehicle enters the track range in time, it is committed to the conflict trajectory and will eventually be able to pass the conflict areas in the trajectory. Otherwise, it needs a new allocation to enter. The allocation request is sent to other vehicles using the **geocast** operation offered by the Vertigo implementation. The result time t_{result}, target time t_{target}, and start time t_{start} are each set to $t_{now} + \delta_{request}$ where $\delta_{request}$ is the length of the time frame of the request, which is configurable per scenario, but the same for all vehicles. The end time is set to $t_{now} + \delta_{request} + \delta_{commit}$, giving the vehicle a configurable time window of length δ_{commit} to enter the conflict trajectory. Delivery time is not used.

The target area A_{target} of the query is selected to be big enough to include all vehicles that may be competing for an allocation of the same conflict area before t_{result} when vehicle i processes the message, or already have an allocation. The initial (range-based) area consists of the set of all track ranges in the conflict areas in T_i. Vehicles in a conflict area certainly have an allocation for the conflict area, since they would not have entered otherwise. Vehicles in front of the conflict areas (in the opposite direction of travel) may have obtained or requested an allocation.

The required target area is bounded by the fact that vehicles may only request an allocation within z meters of a conflict area. However, it also needs to be considered that when a vehicle constructs a conflict trajectory, it may pass through multiple conflict areas in search of conflict-free space. It is therefore possible for vehicles to request an allocation for a conflict area that is more than z meters away, if the conflict area is less than $l_i+d_{min}+l_{commit}$ away from other conflict areas. The solution is to expand the initial area by $l_i+d_{min}+l_{commit}$ in the opposite direction of travel using an expansion algorithm similar to the one described in Section 4.5.1, but for track ranges. The expanded area may intersect with conflict areas that were not in the original conflict trajectory. In that case, the expansion is repeated from the set of track ranges in conflict areas that overlap with the expanded area, until the expansion no longer adds any new conflict areas.

The area obtained by taking the set of track ranges in the conflict areas is expanded by z to include all vehicles that might be competing for one of the conflict areas at the same time. Since some vehicles might start a query too early due to their position inaccuracy, a further expansion of $\alpha_{position}$, representing the worst-case position inaccuracy vehicles might have, is made. Finally, the Vertigo implementation will expand the target area to a delivery area by correcting it for the time until t_{target}, which is $\delta_{request}$. The final set track ranges in the conflict areas in C is therefore expanded by $z+v_{max}\times\delta_{request}+\alpha_{position}$ in the opposite direction of travel to obtain the delivery area containing all vehicles that might have an allocation or might be competing for an allocation of one or more of the conflict areas in T_i. Figure 5.5 shows an example of a delivery area (blue) expanded from a set of conflict areas in the middle of the intersection.

5.5.5 Receiving an Allocation Request

When a vehicle j receives an allocation request from vehicle i containing conflict trajectory T_i through the **receive** event offered by the Vertigo model, j can respond with one of three answers: `accept`, `reject`, or `tentative`. An `accept` response means the receiver will permit the sender to proceed with entering into T_i in the specified time window, a `reject` response means it will not. A `tentative` response means the receiver will allow the sender to proceed after the receiver has passed the conflict areas in T_i, or released its own allocation. A `tentative` response also contains the sequence number

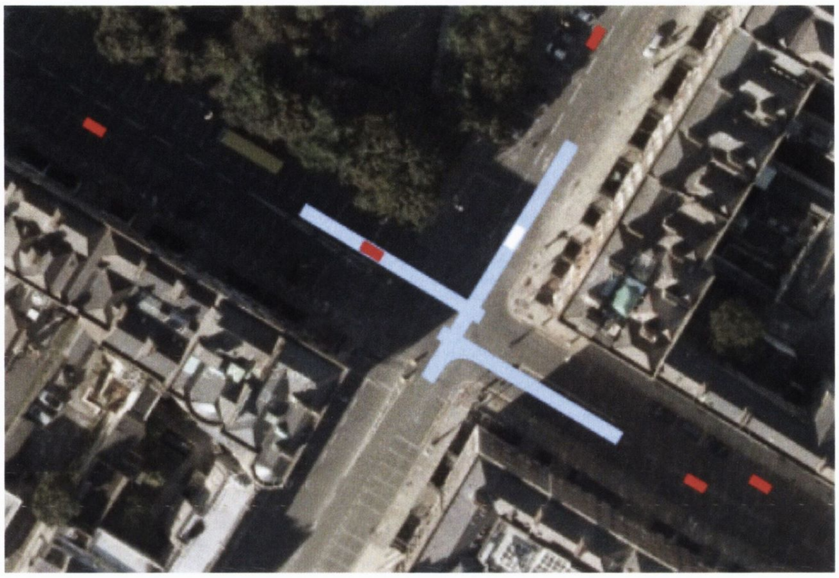


Fig. 5.5: Delivery area (blue) expanded from conflict areas

of j's current allocation, such that the i can find the conflict trajectory T_j from a prior allocation request by j. If i never received such a request, a `tentative` response is treated as a `reject` response.

Which response vehicle j sends depends to an allocation request r on its own allocation a. To choose a response, vehicle j applies the following logic:

```
given: $a \in$ Allocation, $state \in$ AllocationState.
func chooseResponse($r \in$ AllocationRequest).
if $state =$ initial $\vee state =$ released then
  return (accept).
end if
if $|a.C \cap r.C| = 0$ then
  return (accept).
end if
if $state =$ pending then
```

```
        if $a.t_{result} < r.t_{result}$ then
            return (reject).
        else if $a.vehicle_id < r.vehicle_id$ then
            return (reject).
        end if
        $state \leftarrow$ released.
        return (accept).
    end if
    if $state =$ obtained $\vee state =$ committed then
        return (tentative,$a.sequence\_number$)
    end if
    end chooseResponse.
```

A few details such as the definition of allocations are omitted here. C gives the set of conflict areas in the conflict trajectory of the allocation (request). If vehicle j has no allocation or the set of conflict areas in its allocation does not intersect with the set of conflict areas in the received allocation request, the receiver sends an `accept` response to vehicle i. Otherwise, if j's allocation is already in state `obtained` or `committed`, it sends a `tentative` response to i, creating a dependency of i's allocation on j's allocation. In the case where j is in the process of requesting an allocation (in state `pending`) for one or more of the same conflict areas, the vehicle whose allocation has the lowest t_{result} wins, or otherwise the vehicle with the lowest identifier (in $vehicle_id$) wins. If j is this vehicle, it sends a `reject` response to i. Otherwise, it sends an `accept` response and releases its own allocation. In both cases, i will make the opposite decision.

5.5.6 Obtaining an Allocation

A vehicle i that sends an allocation request can only continue into conflict trajectory T_i if *all* vehicles respond with either an `accept` or `tentative` response. At time

t_{result}, the Vertigo implementation passes back the set of receivers R and membership tuple $\mathcal{M}(M, A, t)$ back to the application in a **result** event. If it is true that $R \supseteq M \land A \supseteq A_{target}$ (no omission failures), no `reject` responses were received, and the vehicle has previously received allocation requests for the sequence numbers contained in `tentative` responses, then the allocation request succeeded and the allocation moves to state `obtained`. The state `obtained` gives vehicle i permission to enter into the conflict trajectory T_i between t_{start} and t_{end}.

5.5.7 Spatial Commit

The state of an allocation proceeds to `committed` when a vehicle i moves into the commit area of conflict trajectory T_i between t_{start} and t_{end}. Vehicle i must ensure that it is unambiguously inside (overlapping with) or outside (not overlapping with) the conflict trajectory at t_{end}, considering the inaccuracy of positioning sensors, its current speed, and ability to accelerate and decelerate. Once an allocation is committed, the vehicle will eventually be able to move to the end of the conflict trajectory. If vehicle i fails to enter the conflict trajectory before t_{end}, because it did not maintain enough speed for instance, then the allocation is moved to state `released` and vehicle i must restart the coordination protocol to obtain a new allocation. To simplify verification, the vehicle must wait at least another $\delta_{request}$ before restarting.

Vehicle i moving into the conflict trajectory can be seen as a transaction. At t_{end}, vehicle i is either inside or outside the area. It is the responsibility of vehicles whose allocation depends on i's allocation to check whether or not i is in the conflict trajectory.

5.5.8 Resolving Allocation Dependencies

Once in the commit area of a conflict trajectory, a vehicle i computes the allocated distance, which is the length of the sequence of track ranges in the conflict trajectory for which all dependencies have been resolved for all conflict areas. A conflict area c has an unresolved dependency if at least one of the vehicles that sent a `tentative` response has yet to pass the conflict area, based on its last-known position obtained from beacons.

To compute the allocated distance, vehicle i iterates through the track ranges in the conflict trajectory and for each track range it determines whether there is a conflict area with an unresolved dependency, in which case the sequence stops at the start of the current track range. To determine whether there is a conflict area with an unresolved dependency, the vehicle iterates through every (not yet processed) conflict area c and looks up its dependencies, which are allocation requests whose sequence number was contained in a tentative response from a vehicle j and whose conflict trajectory T_j also passes through the conflict area c.

Vehicle i makes several attempts to resolve dependencies for a conflict area c using information from the beaconing service and the membership service. If a position beacon has been received from vehicle j showing that j was outside of the conflict trajectory after t_{end}, the dependency is resolved. Either vehicle j made it past the conflict trajectory or it failed to enter in time. In either case, it is no longer a dependency for any conflict area that vehicle i is about to enter. If a recent position beacon is available that shows the vehicle is still on the conflict trajectory, the set of conflict areas C_j that still lie ahead of vehicle j in its conflict trajectory is computed. The dependency is resolved if $c \notin C_j$, and unresolved otherwise.

In a worst-case communication scenario, vehicle j might have already left the area without communicating its latest position. In that case, the membership service can be used to confirm the absence of vehicle j from the conflict trajectory. If a tuple $\mathcal{M}(M, A, t)$ can be found for which $A \supseteq T_j$ and $j \notin M$, confirming that j is no longer in T_j, the dependency is resolved. The membership service allows the system to progress, even in the presence of unbounded omission failures of communication from vehicle j to i.

The length of the sequence of track ranges for which all dependencies have been resolved or discarded is used as a virtual distance measurement when determining the speed of a vehicle (though a shorter, real distance measurement may override it). An unresolved dependency thus acts as a "brick wall", which would cause the vehicle to have to stop. In practice, this translates into the vehicle waiting for another vehicle to pass, and accelerating once it has. Thus, a common behaviour for vehicles is achieved through a series of logical steps.

Once vehicle i passes the last conflict area in the conflict trajectory, its allocation is moved to state `released` and the vehicle can return to regular driving until it encounters another conflict area.

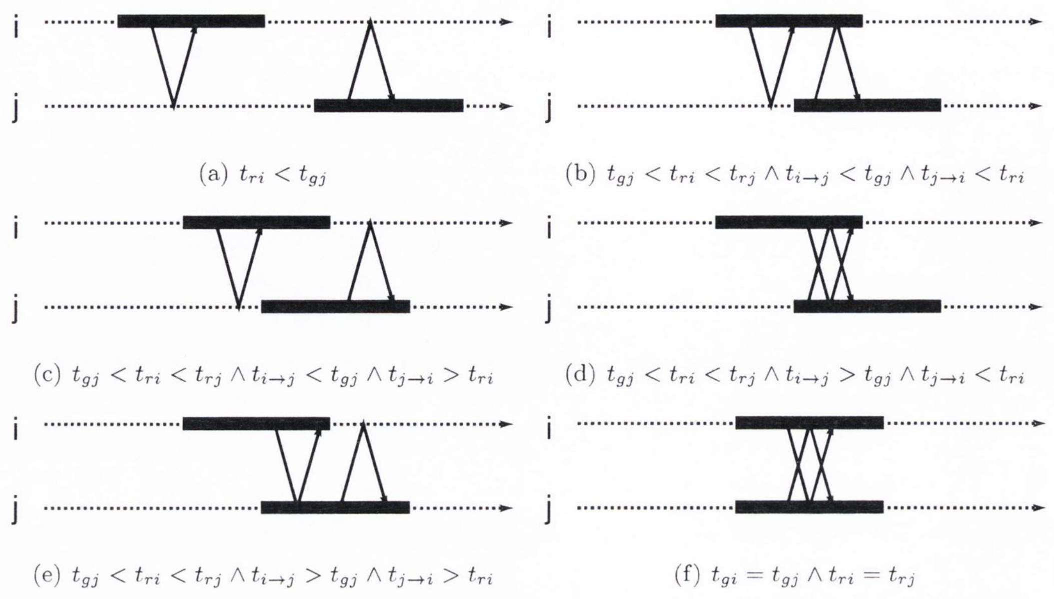
5.6 Formal Analysis

The main concern of the coordination protocol is to guarantee mutual exclusion with regards to conflict areas: no two vehicles can be in the same conflict area at the same time. The way the protocol achieves this is by creating an ordering between allocations of a particular conflict area. The ordering is represented as a distributed dependency graph between allocations that is constructed through tentative responses. A vehicle may not enter a conflict area until it has confirmed that all of the vehicles on which its allocation depends have either passed the conflict area or released their allocation.

No two vehicles may obtain an allocation of a conflict area without an ordering being defined between their allocations. This is achieved by ensuring that any vehicle requesting an allocation needs confirmation from *all* vehicles that might be requesting an allocation of the same conflict area at the same time, or have already obtained or committed to an allocation of the conflict area. The delivery area is expanded to be big enough to include all these vehicles as described in Section 5.5.4.

The remainder of this section gives a proof by exhaustion showing that no two vehicles i and j that are in each other's delivery area can have an allocation in state `obtained` or `committed` for the same conflict area at the same time without an ordering being defined between them.

Let vehicle i send an allocation request at t_{gi} (**geocast** event) with the result deadline



(a) $t_{ri} < t_{gj}$

(b) $t_{gj} < t_{ri} < t_{rj} \wedge t_{i\to j} < t_{gj} \wedge t_{j\to i} < t_{ri}$

(c) $t_{gj} < t_{ri} < t_{rj} \wedge t_{i\to j} < t_{gj} \wedge t_{j\to i} > t_{ri}$

(d) $t_{gj} < t_{ri} < t_{rj} \wedge t_{i\to j} > t_{gj} \wedge t_{j\to i} < t_{ri}$

(e) $t_{gj} < t_{ri} < t_{rj} \wedge t_{i\to j} > t_{gj} \wedge t_{j\to i} > t_{ri}$

(f) $t_{gi} = t_{gj} \wedge t_{ri} = t_{rj}$

Fig. 5.6: Scenarios for concurrent allocation requests by vehicles i and j

set to t_{ri} (**result** event), and vehicle j send an allocation request at t_{gj} with the result deadline set to t_{rj}. Without loss of generality, assume $t_{ri} \leq t_{rj}$. Since $\delta_{request} = t_{result} - t_{geocast}$ is the same for all vehicles, and vehicles must wait at least $\delta_{request}$ when restarting the protocol after a failed commit, the time frames of the allocation requests of i and j can only overlap on one side. Vehicle i could not start another query before t_{rj} if for i's first query $t_{ri} > t_{gj}$. This leads to small number of possible cases with regards to the order of events. In the special case where $t_{ri} = t_{rj}$, the unique identifiers of i and j determine priority, whereas in all other cases i takes priority, being the vehicle with the lower t_{result}.

The time at which the **receive** event for the allocation request from j occurs at i is denoted $t_{j\to i}$, and the time at which the **receive** event for the allocation request from i occurs at j is denoted $t_{i\to j}$. Events at a single vehicle are processed atomically, and can therefore not have the same time.

The case in which communication is *not* successful is covered by the fact that positive responses from all vehicles are required for an allocation to proceed to state `obtained`. An omission failure is detectable using Vertigo and will proceed the state machine to state `released`, creating no conflict. Table 5.6 gives all the scenarios for two allocation requests for the same conflict area in case communication *is* successful. For clarity, timelines and communication of vehicles i and j are shown in Figures 5.6 in which the thick line represents the period between $t_{geocast}$ and t_{result}.

Table 5.1: Scenarios for concurrent allocation requests

Let $t_{ri} < t_{gj}$:

Vehicle j in state `initial` at $t_{i \to j}$, responds `accept` to i

- Let vehicle i release the allocation before $t_{j \to i}$:
    - Vehicle i in state `released` at $t_{j \to i}$, responds `accept` to j
    - Such that j may proceed to state `obtained`, and i proceeded to state `released`.
- Otherwise:
    - Vehicle i in state `obtained` or `committed` at $t_{j \to i}$, responds `tentative` to j
    - Such that both may proceed to state `obtained`, but j depends on i.

Let $t_{gj} < t_{ri} < t_{rj}$:

- Let $t_{i \to j} < t_{gj} \wedge t_{j \to i} < t_{ri}$:
    - Vehicle j in state `initial` at $t_{i \to j}$, responds `accept` to i
    - Vehicle i in state `pending` at $t_{j \to i}$, responds `reject` to j
    - Such that i may proceed to state `obtained`, and j proceeds to state `released`.
- Let $t_{i \to j} < t_{gj} \wedge t_{j \to i} > t_{ri}$:
    - Vehicle j in state `initial` at $t_{i \to j}$, responds `accept` to i
    - Let vehicle i release the allocation before $t_{j \to i}$:
        - Vehicle i in state `released` at $t_{j \to i}$, responds `accept` to j

```
            Such that $j$ may proceed to state obtained, and $i$ proceeded to state released.
        Otherwise:
            Vehicle $i$ in state obtained or committed at $t_{j \to i}$, responds tentative to $j$
            Such that both may proceed to state obtained, but $j$ depends on $i$.
    Let $t_{i \to j} > t_{gj} \wedge t_{j \to i} < t_{ri}$:
        Vehicle $j$ in state pending at $t_{i \to j}$, responds accept to $i$, $state_j \leftarrow$ released
        Vehicle $i$ in state pending at $t_{j \to i}$, responds reject to $j$
        Such $i$ may proceed to state obtained, and $j$ proceeded to state released.
    Let $t_{i \to j} > t_{gj} \wedge t_{j \to i} > t_{ri}$:
        Vehicle $j$ in state pending at $t_{i \to j}$, responds accept to $i$, $state_j \leftarrow$ released
        Let vehicle $i$ release the allocation before $t_{j \to i}$:
            Vehicle $i$ in state released at $t_{j \to i}$, responds accept to $j$
            Such that both proceeded to state released.
        Otherwise:
            Vehicle $i$ in state obtained or committed at $t_{j \to i}$, responds tentative to $j$
            Such that $i$ may proceed to state obtained, and $j$ proceeded to state released.
```

```
Let $t_{ri} = t_{rj}$:
    Let $i < j$:
      Vehicle $j$ in state pending at $t_{i \to j}$, responds accept to $i$, $state_j \leftarrow$released
      Vehicle $i$ in state pending at $t_{j \to i}$, responds reject to $j$
      Such that $i$ may proceed to state obtained, and $j$ proceeded to state released.
    Let $i > j$:
      Vehicle $j$ in state pending at $t_{i \to j}$, responds reject to $i$
      Vehicle $i$ in state pending at $t_{j \to i}$, responds accept to $j$, $state_i \leftarrow$released
      Such that $j$ may proceed to state obtained, and $i$ proceeded to state released.
```

In all cases, either no vehicle proceeded its allocation to state `obtained`, only one of them did, or j was made aware of its dependency on i. Given that a `tentative` response is only sent by vehicle i when $t_{ri} < t_{rj}$, an allocation will only ever depend on allocations requests that preceded it. The protocol is therefore free of cycles in the dependency graph and thus free of deadlocks if fully synchronized clocks are assumed. A protocol and proof for partially synchronized clocks (e.g. using TrueTime [Corbett 12]) is beyond the scope of the thesis, but a topic of future work.

5.7 Discussion

The protocol presented in this chapter arbitrates arbitrary intersection crossings using Vertigo. Mutual exclusion with regards to conflict areas is achieved by creating a strict ordering between allocations, by constructing a distributed, cycle-free dependency graph through `tentative` responses. If requests are concurrent, one side will forfeit its allocation and retry later. Safety is preserved in the presence of unbounded omission failures, since they can be detected using the Vertigo communication model. Progress is not guaranteed in the case of unbounded omission failures, since communication may never succeed at all, but the membership service can be used to handle cases where a vehicle leaves an area without further communication.

Chapter 6

Evaluation

In this chapter, the feasibility and applicability of the Vertigo communication model are further demonstrated and characterized. Feasibility is characterized by the conditions under which success can be achieved by the implementation from Chapter 4 in terms of communication and sensing capabilities, and traffic density. Applicability is characterized by the performance of the distributed coordination protocol from Chapter 5 as an intersection management solution compared to traditional approaches. Simulations of the Vertigo implementation and the coordination protocol show that success rates can be achieved that are sufficiently high to enable the coordination protocol to achieve better traffic flow than regular traffic lights, with relatively conservative sensing and communication capabilities.

6.1 Evaluation strategy

The Vertigo implementation cannot be easily studied in isolation. It is meant specifically for cooperative automated driving applications in which vehicles coordinate their driving decisions with each other. The specific situation in which Vertigo is used by the application largely determines how well it performs. The performance of Vertigo depends on the environment, such as the line-of-sight of ranging sensors, as well as the behaviour of the vehicles themselves that lead to certain positions, speeds, and channel utilization.

Generic mobility models cannot be used for evaluation, since they allow overlap between vehicles, which would create impossible sensing situations. Vertigo can only be studied in the context of a functional coordination application that preserves safety. The scenario used to evaluate the Vertigo implementation is a four-way intersection managed using the distributed coordination protocol from Chapter 5.

A series of simulation experiments characterizes the relationship between the operating environment, the success rate of Vertigo, and progress in the scenario using the coordination protocol. The environment in which the Vertigo implementation operates, particularly the available hardware capabilities, can affect its performance in multiple ways. Low transmission power can cause omission failures over longer ranges, while reduced sensor range makes it harder to obtain sufficient membership information. High traffic density can cause the wireless network to become over-utilized. High positioning uncertainty reduces the likelihood of obtaining sufficient membership information. The effects of these factors on success rate and the resulting traffic flow are studied in a set of parameter studies, finding the minimal conditions under which the Vertigo implementation can operate in the intersection scenario.

The challenge for any automated driving system is not just to preserve safety, but to do so while making progress towards certain goals, in this case crossing an intersection. Without progress, safety is trivially achieved as no vehicles are moving. While optimizing traffic flow is not a particular goal of this thesis, the rate of traffic achieved using the coordination protocol is compared to the traditional intersection management approach of traffic lights to demonstrate that Vertigo can achieve success rates high enough to implement a feasible traffic management solution and is thus applicable to this domain.

6.2 Simulator

The evaluation makes use of a novel simulator called roundasim. It is so named because it was originally intended for a roundabout scenario, but no coordination solution is

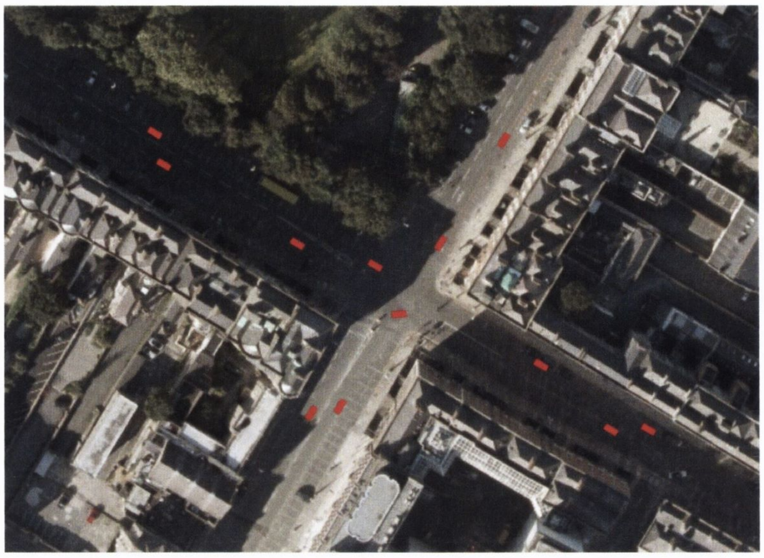


Fig. 6.1: A simulation in roundasim with cars shown in red

currently available for the roundabout. The simulator has since evolved into a general purpose cooperative automated driving simulator with communication, sensors, software services, visualizations (see Figure 6.1[1]), and a tool to draw traffic scenarios with arbitrary roads and buildings. The network simulation library SWANS [Barr 05] has been integrated to provide wireless network simulation. The simulator can be run and viewed using a web browser at `http://thesis.marcoslot.net/`.

A number of alternative simulators were explored for evaluation, but none have the required combination of simulating actuators, sensors, and wireless networking, or lack visualization, scale, or a fine-grained model of time. Microscopic traffic simulators such as VISSIM [Fellendorf 01] and SUMO [Behrisch 11] use fixed time steps, meaning all the cars move forward by a fixed amount of time. This behaviour may be appropriate for evaluating traffic flow, but the fidelity of interactions in cooperative automated driving is lost in the coarseness of the time steps. Conversely, wireless communication happens on a nanosecond scale. Network simulators such as ns-2 [Chen 06] and OPNET [Chang 99]

[1]Map data ©2013 Google

use a queue of events ordered by a discrete execution time, allowing for arbitrarily small time steps. However, they do not model continuous movements.

While the two simulation models are not fundamentally incompatible, synchronization between a fixed time step simulator and a discrete event simulator is complex and error-prone. We have attempted such an integration between OPNET and VISSIM in [O'Hara 12], but it was lacking in performance, flexibility, and visualization, making it a difficult environment in which to write complex distributed algorithms. Visualization is particularly important for viewing the spatial interactions between vehicles when debugging and tuning coordination and control algorithms.

Robotics simulators such as Microsoft Robotics Studio [Jackson 07] typically use a real-time execution model, in which simulation time follows the system clock and can be viewed at a normal speed. Objects are moved forward in iterations based on the system clock time that has passed since the start of the last iteration. The drawback of this model is that the fidelity of the simulator may depend on the performance of the computer and the simulation may break when unable to keep up with real time. PreScan is a simulator for Advanced Driver Assistance Systems [Hendriks 10], with fine-grained simulation of vehicle dynamics and sensors, and visualization tools. It uses fixed time steps at a high granularity. Simulations can run in real time, but degrade gracefully. Unfortunately, PreScan has only very basic simulation of wireless networking, and lacks scalability. Vehicles cannot be added or removed dynamically.

Roundasim is based on the lessons from working with the different types of simulators described (fixed time step, discrete event, real-time). Control logic, simulation logic, sensors, and wireless communication events are executed using a discrete event queue. Before an event is executed, the 'physical' world is moved forward by the amount of simulation time since the last event by recomputing the positions and velocities of all vehicles, based on current speed, acceleration, and trajectory. Alternatively, the simulation can run in a real-time mode to allow visualization. In this case, the simulation state moves forward from a point in time in one continuous block (time in between discrete events) to another based on the system clock time that has passed since the last iteration, executing all events and blocks in between. If the computer is too slow to run the simulation in real-time, this has no negative impact on the fidelity of the simulation itself. However, visualizations may look less smooth.

6.2.1 Driving

Driving in roundasim is simulated through the use of tracks, interconnected 2D line strings that are generated from Bézier curves. The position, speed, and acceleration of a vehicle are defined in one dimension, as an offset on a track. This allows extremely efficient motion simulation by simply increasing the offset, while avoiding the need for explicit steering simulation. The one-dimensional offset can be translated into a two-dimensional position by taking the point and orientation on the line string at the given offset. The point, which represents the back of the vehicle, and the orientation can be translated into a polygon to give the two-dimensional outline of the vehicle. The outline of a vehicle is only computed when the vehicle is observed using another vehicle's ranging sensor or is required for visualization. These techniques make the simulator efficient enough to update the physical world before every event, while still supporting real-time visualization of large-scale scenarios.

Every track starts and ends at a given offset on another track, unless the track is at the beginning or end of a scenario. By default, a vehicle will proceed onto the track to which its current track connects once it reaches the end of its current track. However, the vehicle can also switch to tracks that start ahead of it on its current track. A queue of track identifiers is kept for every vehicle. A vehicle proceeds to a track starting directly ahead of it when the identifier of the track is at the head of the queue. A vehicle can steer itself by pushing track identifiers of upcoming track starts onto the queue.

To move forward, a vehicle i sets its acceleration a_i, which may be negative in case of deceleration. When the state of the physical world is moved forward by Δ, then for a starting speed of v_i, the distance driven is computed as $v_i \times \delta_s + 0.5 \times a_i \times \delta_s^2$, where

$\delta_s = \mathbf{max}(\Delta, (v_{max} - v_i)/a_i, v_i/-a_i)$, since the acceleration only continues until the vehicle reaches the maximum or minimum speed after which acceleration becomes 0. After the update, the new speed of the vehicle is $v_i + a_i \times \delta_s$.

6.2.2 Sensors

Roundasim implements the sensor interfaces from Appendix B for **position**, **orientation**, and **ranges** (LIDAR). Position and orientation values can be taken directly from the state of the simulation or computed from the current position on a track. A vehicle has a set of ranging sensors defined, each with a specific position and orientation on the vehicle, and a given number of beams, angular resolution, and range. The layout used in experiments is the one shown in the example in Figure 4.4 in Chapter 4. Each vehicle has 4 sideways single-beam sensors on the sides and two 180° sensors on the front and back. Simulation of ranging sensors is performed through ray casting, searching for intersections between line segments that represent LIDAR beams and the outlines of other vehicles or buildings. The outlines of vehicles are rectangles of 4.12 by 1.83 meters (frequently mentioned, though non-reputable values for the average length and width of cars). The simulator allows buildings to be defined as arbitrary polygons. Ranging sensors are simulated every 100ms. Delay between different rays or sensors is not currently simulated.

6.2.3 Wireless ad-hoc network

The wireless network in roundasim is simulated using SWANS [Barr 05] (Scalable Wireless Ad-hoc Network Simulator), a Java library for simulating an ad-hoc 802.11 network. It is normally used in combination with JiST, a virtual machine simulator. The JiST execution model used by SWANS is not entirely compatible with a discrete event simulator that uses explicit scheduling, but the modifications to make SWANS compatible with roundasim are minor. The parameters used to configure SWANS are taken from the 802.11p standard for wireless access in vehicular environments [IEEE 10]. Rayleigh

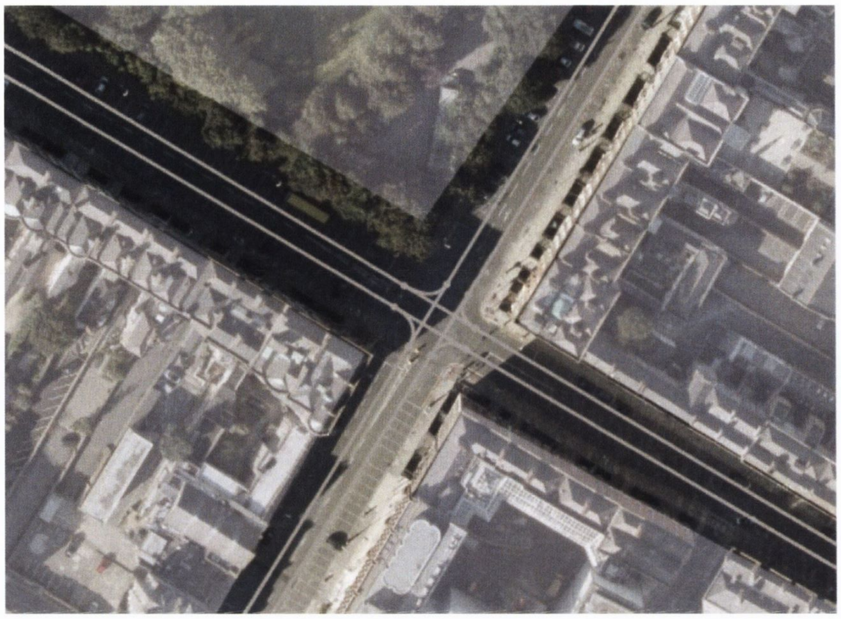


Fig. 6.2: Tracks and buildings on the Merrion Square intersection shown in gray

fading and two-ray propagation [Proakis 08] are used to model the physical channel. Control software running in the simulator uses SWANS through the generic **unicast**, **broadcast**, and **receive** interfaces defined in Appendix B.

6.2.4 Scenario

The scenario used in the experiment is modelled after a four-way intersection near Merrion Square in Dublin, Ireland. In the simulated scenario, tracks are defined for the roads, left turns, and crossings, as shown in Figure 6.2. No right turns are defined to allow for comparison between vehicles using Vertigo for coordination and vehicles that only drive based on forward-looking distance and traffic lights. Without coordination, vehicles turning right might collide with vehicles driving in the opposite direction.

In the regular scenario, vehicles use Vertigo and the coordination protocol to cross the intersection. The results of this scenario are compared to one in which traffic lights are used to manage traffic. The traffic lights switch between the East-West and North-South directions. The phases of the traffic lights are 60 seconds on green, 3 seconds on orange, and 2 seconds on double red, before switching directions. These values are based on the actual traffic lights on the Merrion Square intersection, and therefore represent the traffic management solution in use today.

Vehicles are added to the scenario at the start of tracks going towards the intersection. The intervals between new vehicles being added follows a Poisson distribution with a configurable rate. Whenever a vehicle is added, the direction from which a vehicle arrives and the direction it takes on the intersection (left or straight) are chosen randomly. If another vehicle is close to the entry point, such that the minimum distance between vehicles would be violated by adding a vehicle, then the new vehicle is omitted. The actual entry rate may therefore be lower than the *offered* entry rate, as some vehicles are omitted. When the intersection is at maximum capacity, queues will form in front of it, and vehicles are only added when there is enough room in the queue. The actual rate of entry will therefore be roughly the same as the rate of departure.

6.2.5 Controller

The main purpose of the simulator is to evaluate a (cooperative) automated driving controller, a piece of software running on a computer inside the vehicle that can access its sensors, actuators, and networking equipment. The controller can set the acceleration and the sequence of tracks to follow and can use the sensors, wireless network, and road map. Two controllers are used in the evaluation. The first one uses the implementations of the coordination protocol and Vertigo defined in the previous chapters to guarantee mutual exclusion for conflict areas. The second controller ignores conflict areas, but only enters the intersection during the time interval in which the traffic light for its direction is green. Both controllers use the Intelligent Driver Model [Treiber 13] to decide on acceleration.

6.3 Experiments

This section describes the experiments performed to evaluate the Vertigo implementation and the coordination protocol. Metrics of key interest are the average success rate of Vertigo queries, the network overhead as the average bitrate of transmissions, the average rate of traffic, and the time vehicles need to cross the intersection as a distribution. These metrics are studied across multiple runs while varying one of the environment parameters that could affect the performance of the Vertigo implementation. The parameters studied in the experiments are the transmission power of the WLAN adapter, the range of the LIDAR sensors, the inaccuracy of the position sensor, and the rate at which traffic enters the scenario. While varying one parameter, the other parameters are set to the default values listed in Table 6.1.

The default LIDAR range of 30m is chosen as it is the advertised range of the SICK LMS 291 that was commonly used in the DARPA Grand Challenge in 2005 [Buehler 07], which is what the simulated LIDAR model is based on. More modern LIDARs, such as the Velodyne LIDARs used in the DARPA Urban Challenge in 2008 [Buehler 09], can work reliably at ranges of 60m or more, but a more conservative range is sufficient. The default position inaccuracy of 1.5m is based on the inaccuracy bounds of the combined GPS and intertial positioning system by [Huang 06]. Both the LIDAR range and the position inaccuracy are deliberately based on older types of sensors to show the feasibility of achieving sufficiently high success rates using less capable sensors. The default entry rate of 20 vehicles per minute is low enough not to overload the intersection, but high enough to have a significant number of vehicles cross the intersection in an hour of simulation time (over 1000). The WLAN transmission power is based on the recommendation that was found experimentally, as described in Section 6.3.4.

The coordination protocol has a number of configurable parameters, listed in Table 6.1 (z, b, $\delta_{request}$, δ_{commit}, v_{max}, a_{max}, and a_{min}). The values are chosen in conjunction with each other. For example, if a vehicle drives at a speed of 20m/s (v_{max})

and starts the coordination protocol at a distance of 25m (z) from a conflict area waits 200ms ($\delta_{request}$) to decide whether to continue, it has a distance of 21m to the conflict area at t_{result}, which would be sufficient to stop at a speed of v_{max} and a deceleration of 10m/s^2 (a_{min}). The time between t_{end} and t_{start} of 1.3s gives the vehicle enough time to reach the conflict area before t_{end}. The beaconing rate of 5Hz (b) follows from the fact that all vehicles will have beaconed their membership information at least once between $t_{geocast}$ and t_{result}. Thus, this particular combination of application-specific parameters allows vehicles to cross the intersection at maximum speed, with minimal overhead in terms of beaconing and the size of the delivery area.

Table 6.1: Default values for simulation parameters

Parameter	**Value**
LIDAR range	30m
Position inaccuracy	1.5m
Entry rate	20 vehicles / minute
WLAN transmission power	10dBm
Start distance for coordination protocol (z)	25m
Beaconing rate (b)	5Hz
Time between t_{result} and $t_{geocast}$ ($\delta_{request}$)	200ms
Time between t_{end} and t_{start} (δ_{commit})	1300ms
Maximum speed (v_{max})	72km/h or 40m/s
Maximum acceleration (a_{max})	10m/s^2
Maximum deceleration (a_{min})	-10m/s^2

Simulations are run in blocks of 1 hour of simulation time and repeated for 5 different random number generator seeds. Blocks of 1 hour are chosen primarily as a number that allows for hundreds of Vertigo queries per run, while not taking so long to simulate that it becomes infeasible to do many different runs. The experiments for LIDAR range, position inaccuracy, and entry rate are repeated for comparison to the traffic lights scenario.

6.3.1 LIDAR range

The maximum distance that can be measured reliably using ranging sensors directly affects the membership area that can be obtained. In general, it is expected that a higher maximum range leads to better results. If the maximum range is too short, vehicles cannot cover the target area of geocasts with their LIDAR beams, which would prevent any progress. The maximum range is evaluated by cutting off simulated LIDAR beams at the given distance. The effects of the maximum range on the performance of Vertigo and the coordination protocol are studied by varying it from 10m to 60m in steps of 5m in $11(steps) \times 5(seeds)$ simulation runs. In addition to the empty areas that are used for membership, the maximum LIDAR range also affects the regular forward-looking distance of vehicles. The results from the traffic light scenario are provided to show the extent to which regular driving is affected by a short maximum LIDAR range.

An average success rate of Vertigo queries of 70% or higher is achieved when the maximum LIDAR range is over 30m as shown in Figure 6.3. Below this range, many queries fail, which drives up the channel overhead as the number of vehicles waiting increases and queries need to be retried. A reduced rate of traffic can be sustained with maximum LIDAR ranges as low as 15m. At low LIDAR ranges, vehicles become unable to cross the intersection by themselves as their LIDAR does not cover the target area of a geocast, even when standing near the intersection. However, a vehicle i can still make progress if there are other vehicles on or near the intersection whose LIDAR beams cover the remainder of the target area of i's queries, because the empty areas generated from their LIDAR data will be merged into vehicle i's membership view. Below 15m, vehicles can no longer cover the full intersection and no progress can be made. The traffic light scenario is also affected in this case, since vehicles can only move very slowly when only able to look 10m ahead (minus 1.5m position inaccuracy and 1m LIDAR inaccuracy as

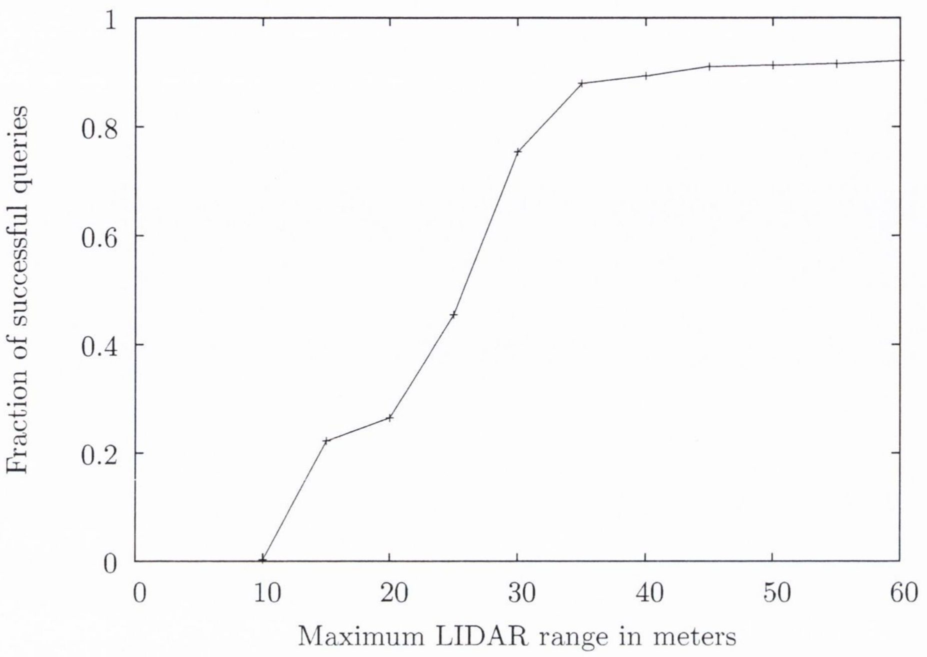


Fig. 6.3: The average success rate of Vertigo queries for different LIDAR ranges

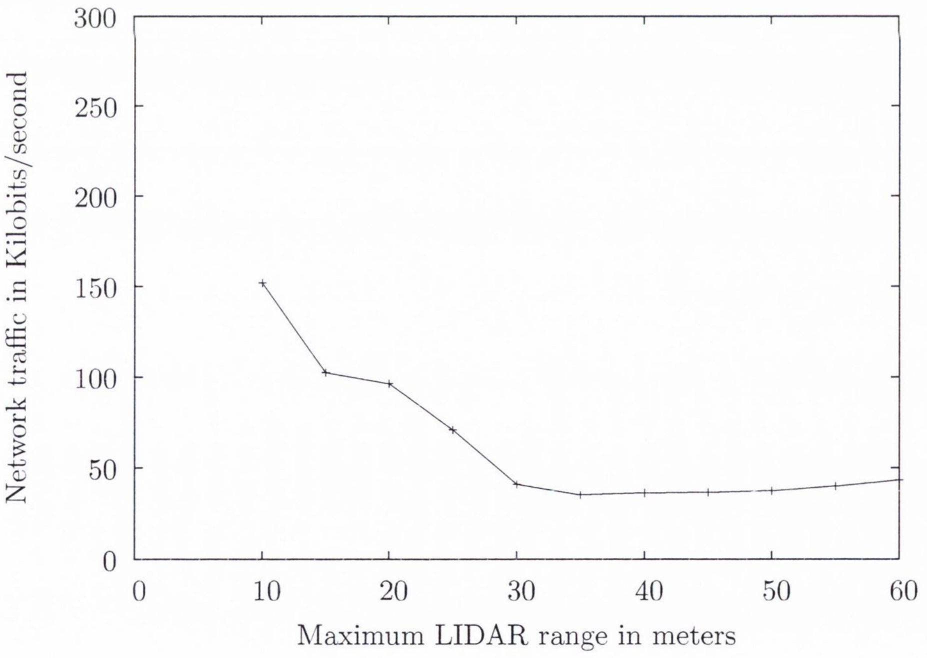


Fig. 6.4: The average network overhead (over all vehicles) for different LIDAR ranges

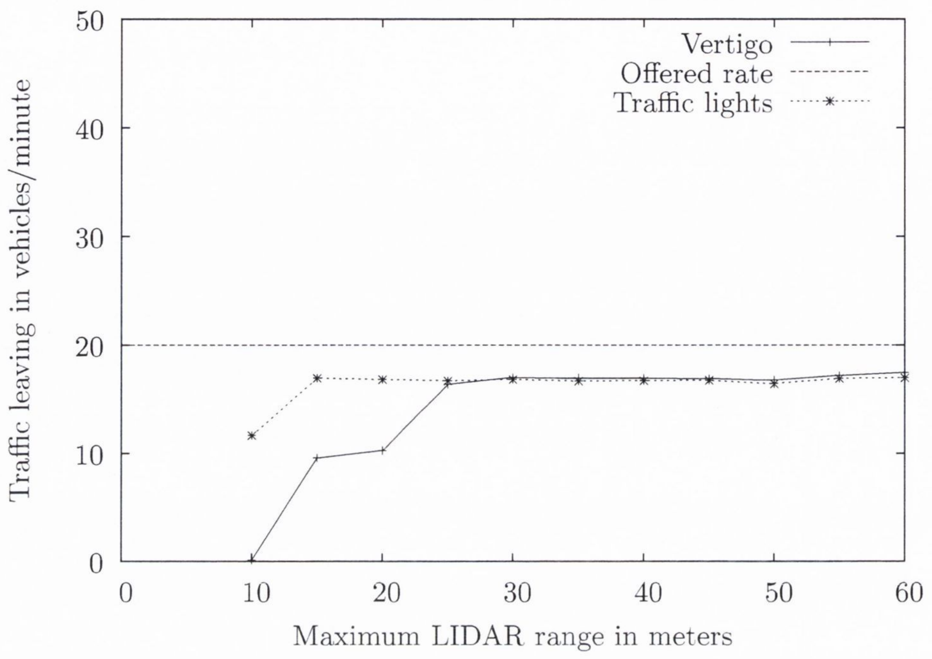


Fig. 6.5: The average rate of traffic over the intersection for different LIDAR ranges

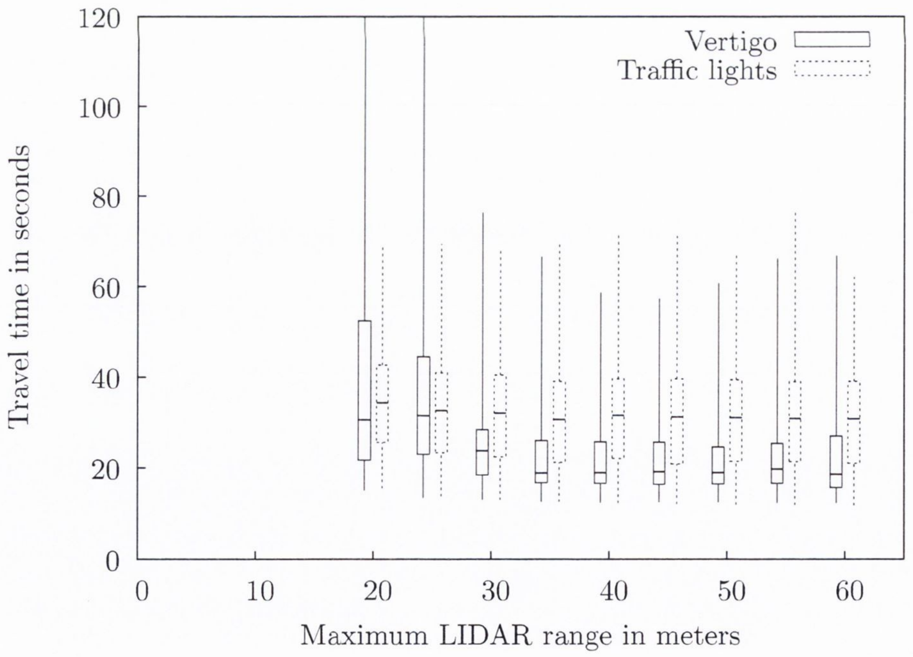


Fig. 6.6: Quartiles of the travel time over the intersection for different LIDAR ranges

discussed in Section 4.4.1.2).

At the default LIDAR range of 30m, the median travel time of vehicles using Vertigo and the coordination protocol is $\sim 23\%$ lower compared to traffic lights and as much as 40% for the most optimal maximum range of 35m. Importantly, the variance in travel times is also much lower when using Vertigo and the coordination protocol. This shows the effectiveness of the coordination protocol as a traffic management solution and the applicability of Vertigo to that domain.

An unanticipated result is that worst-case travel times increase slightly for greater LIDAR ranges as shown in Figure 6.6, despite an increase in success rate and traffic rate. One drawback of a greater maximum LIDAR range is that it requires a greater number of road segment boundaries to represent. The network overhead shown in in Figure 6.4 is roughly 20% higher when the maximum LIDAR range is 60m, when compared to a maximum LIDAR range of 35m. However, this effect is not significant enough to explain travel time increases, as it does not cause queries to fail. The actual reason for the slight travel time increase is vehicles querying too early. If a vehicle i is approaching the intersection without any vehicles ahead of it, it opportunistically sends an allocation request for a conflict trajectory as soon as it is within 25m of a conflict area with the end time for entering the conflict trajectory t_{end} set to $t_{now} + 1.5s$, where t_{now} is the time at which i sends the allocation request. Unless vehicle i can maintain a speed of 60km/h or greater, it will not reach the commit area in time to commit to the allocation. For lower LIDAR ranges, queries made early are more likely to fail, because the LIDAR sensors of the vehicle itself cannot cover the target area of the geocast by themselves. However, those vehicle can retry quickly, since their allocation is released at t_{result}. Vehicles whose query succeeds, but fail to reach the commit area in time, do not retry until after t_{end} and have to stop and wait for a new allocation request to complete successfully. This problem can be solved by a more optimal application that takes speed into account when deciding whether to make a query.

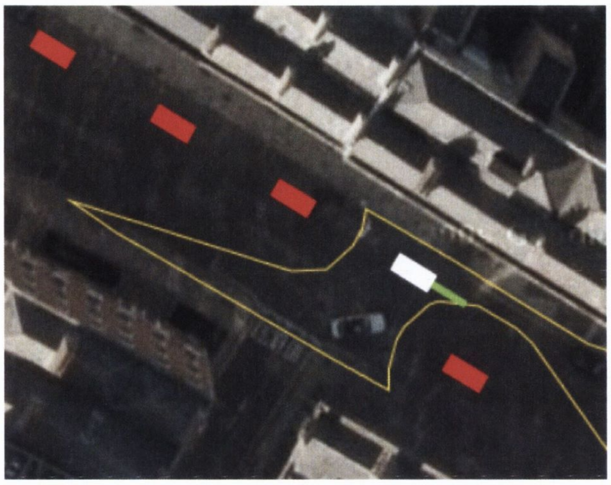


Fig. 6.7: An example of position inaccuracy being too high to continue

6.3.2 Position accuracy

The algorithm for computing the empty area from which membership tuples are derived takes a worst-case bound on position inaccuracy into account by reducing the size of the empty area as described in Section 4.4.1.2. The gaps that vehicles create in each other's empty areas will be bigger as position inaccuracy grows, to the point where the empty areas measured by vehicles no longer overlap. When this happens, the membership area, constructed by decaying and merging the empty areas from neighbours, may have gaps that lie in the target area of the geocast, in which case success cannot be confirmed.

The effects of the position inaccuracy bound are studied by varying it from 0m to 5m in steps of 0.5m in $11(steps) \times 5(seeds)$ simulation runs. The average success rate of Vertigo queries across all runs for a given inaccuracy bound is shown in Figure 6.8. The success rate remains relatively stable until inaccuracy goes over 2m. Above 2m of inaccuracy, the success rate heavily deteriorates. At the same time, the network overhead shown in Figure 6.9 more than doubles up to 125Kbps, as vehicles keep retrying queries in order to obtain an allocation. While the success rate drops to 43% for an inaccuracy bound of 3m, the traffic flow is almost unaffected. Vehicles successfully retry queries,

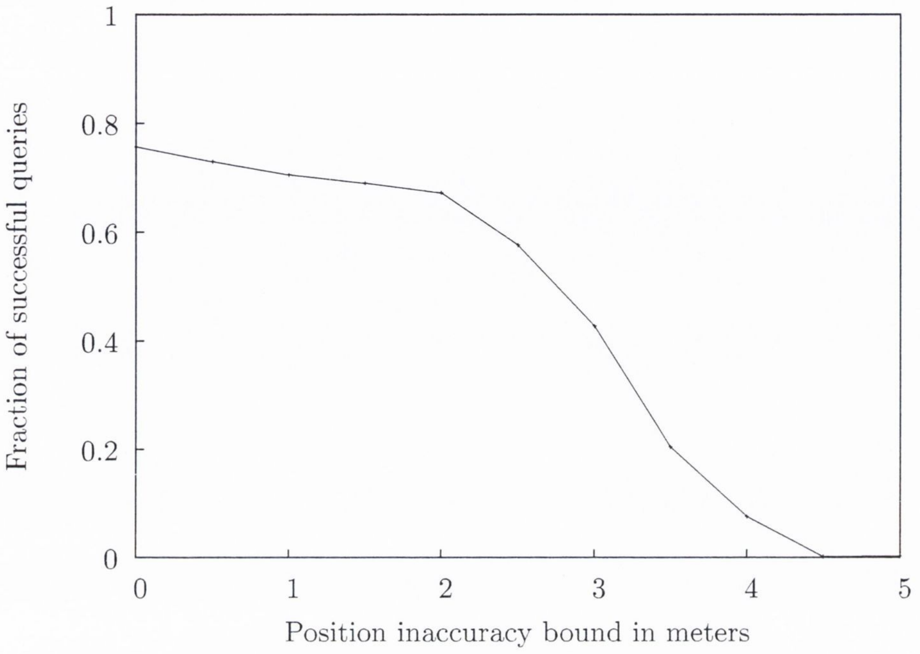


Fig. 6.8: The average success rate of Vertigo queries for different position inaccuracies

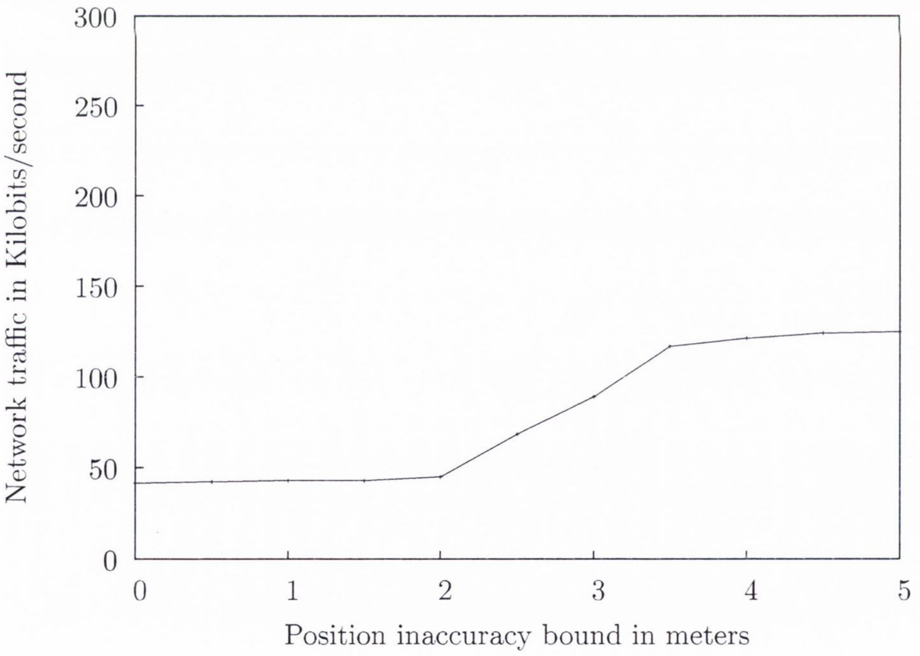


Fig. 6.9: The average network overhead (over all vehicles) for different position inaccuracies

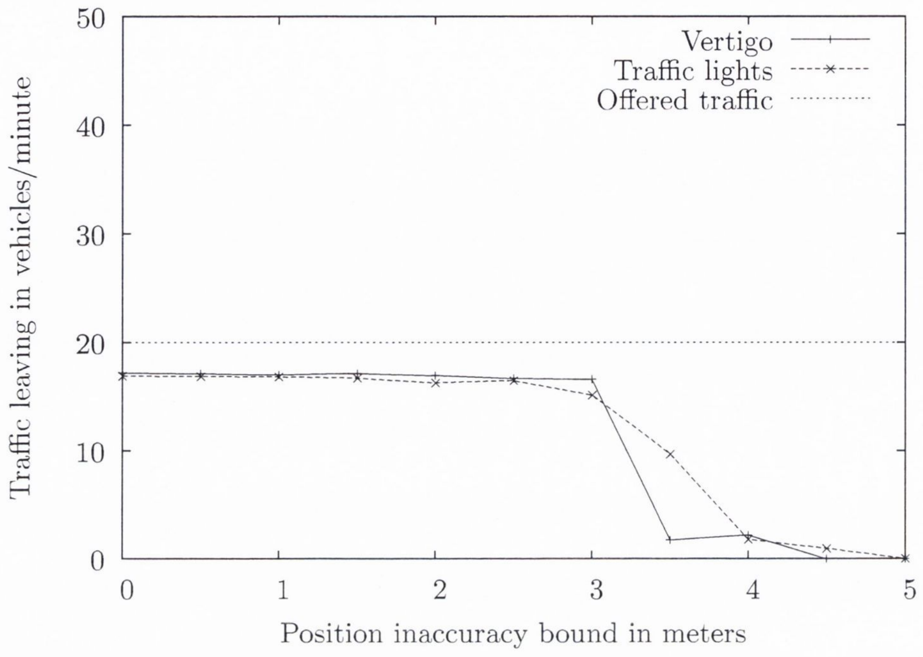


Fig. 6.10: The average rate of traffic for different position inaccuracies

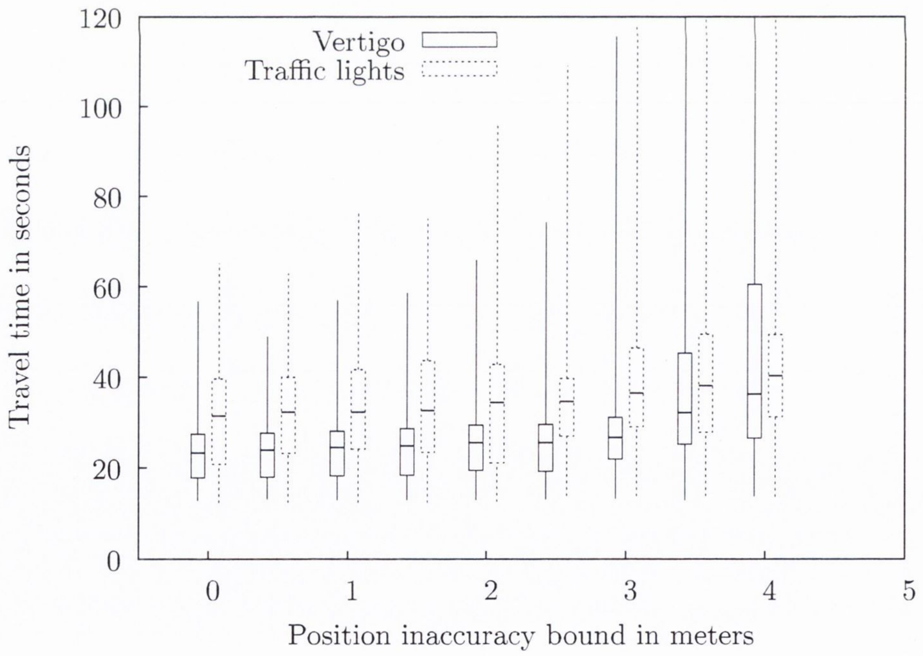


Fig. 6.11: Quartiles of the travel time for different position inaccuracies

allowing them to make progress despite a low success rate. However, travel times increase as vehicles may need multiple attempts to obtain an allocation.

At 4.5m position inaccuracy or more, no progress is made due to a combination of two factors. The position inaccuracy is so high that the membership area always contains gaps and no queries can succeed, thus no progress can happen. However, vehicles do not make any progress in the traffic lights scenario either. The reason is that when a vehicle i measures an empty area, other vehicles create gaps that are large enough to reach the track vehicle i is on as shown in Figure 6.7. The yellow line is the empty area after correcting it for inaccuracy. The green line the empty trajectory derived from the empty area. The empty trajectory is cut short by the gap in the empty area created by a vehicle driving in the other direction and the inaccuracy about its precise whereabouts. The short forward-looking distance causes vehicles to no longer deem it safe to drive forward. This problem is specific to the use of the 'empty trajectory' by the application, but can be generalized by making the observation that the accuracy has become too low for vehicles to distinguish between lanes. Automated vehicles require positioning that is at least accurate enough to follow lanes according to [de Nooij 10], who define an inaccuracy bound of 10cm.

6.3.3 Offered rate of entry

Traffic density affects Vertigo's performance in various ways, such as the number of simultaneous queries, the number of receivers and members, the number of line-of-sight obstacles, and the ability of the coordination application to obtain approval for its intentions. To study the effects of traffic density, the offered rate of entry is varied from 10 vehicles/minute to 100 vehicles/minute in steps of 5 in $19(steps) \times 5(seeds)$ simulation runs. As previously explained, the actual rate of entry converges to the rate of departure, due to the fact that additional vehicles cannot be added when there is a long queue of vehicles in front the intersection.

Perhaps the most important observation when increasing the entry rate is the point

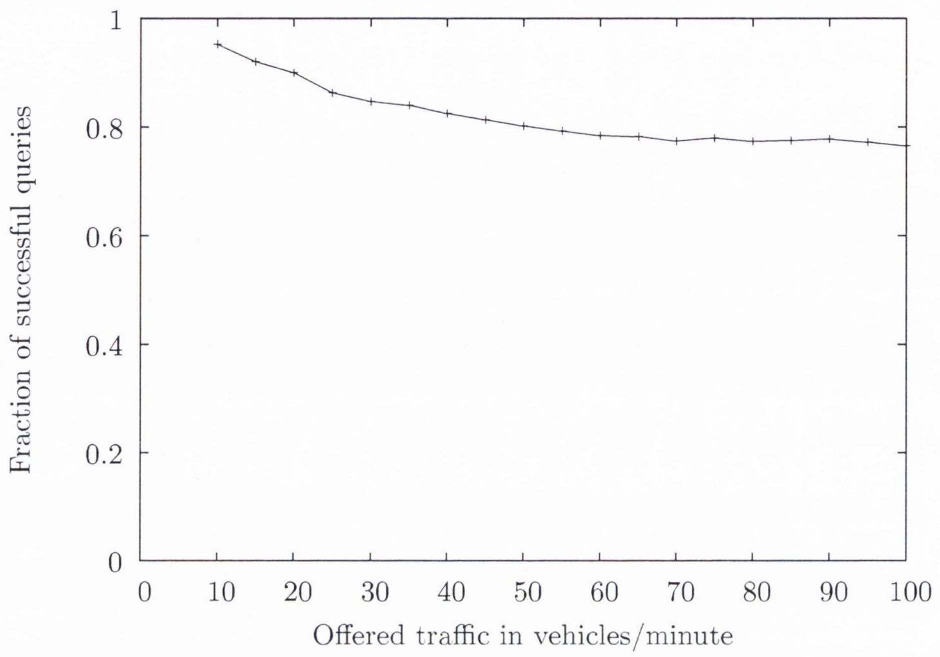


Fig. 6.12: The average success rate of Vertigo queries for different entry rates

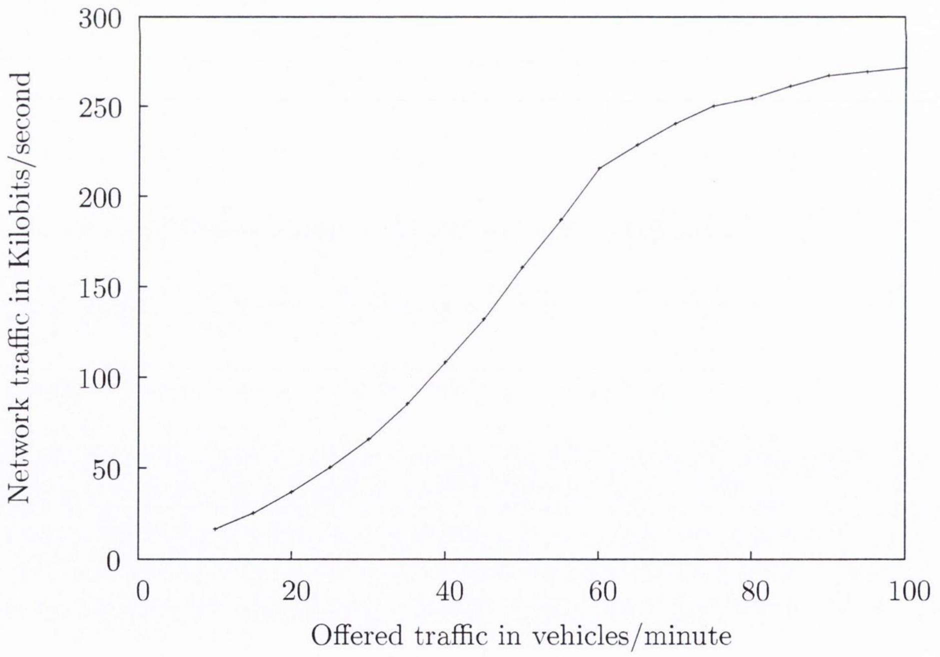


Fig. 6.13: The average network overhead (over all vehicles) for different entry rates

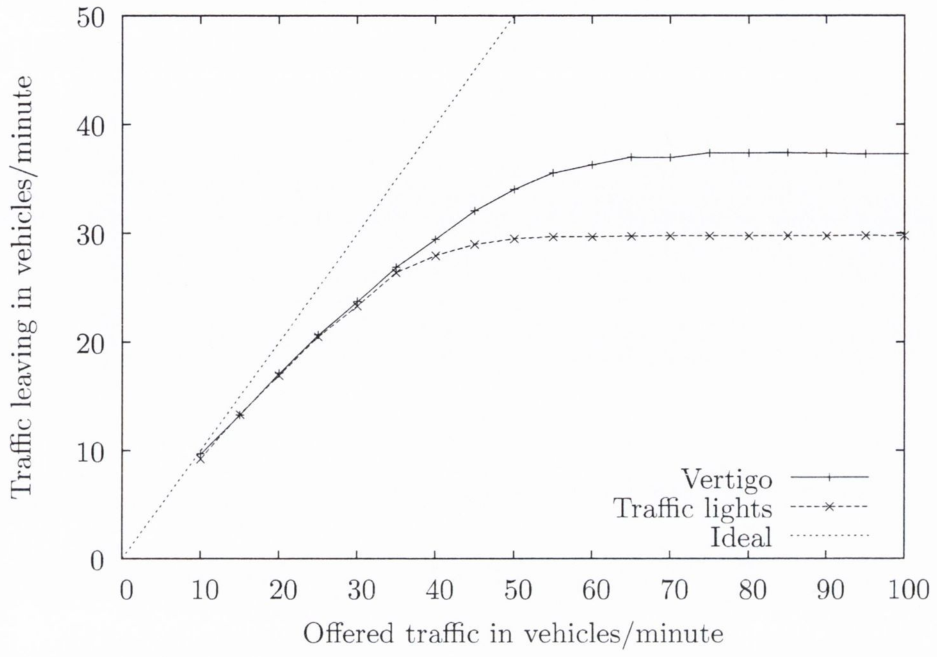


Fig. 6.14: The average rate of traffic over the intersection for different entry rates

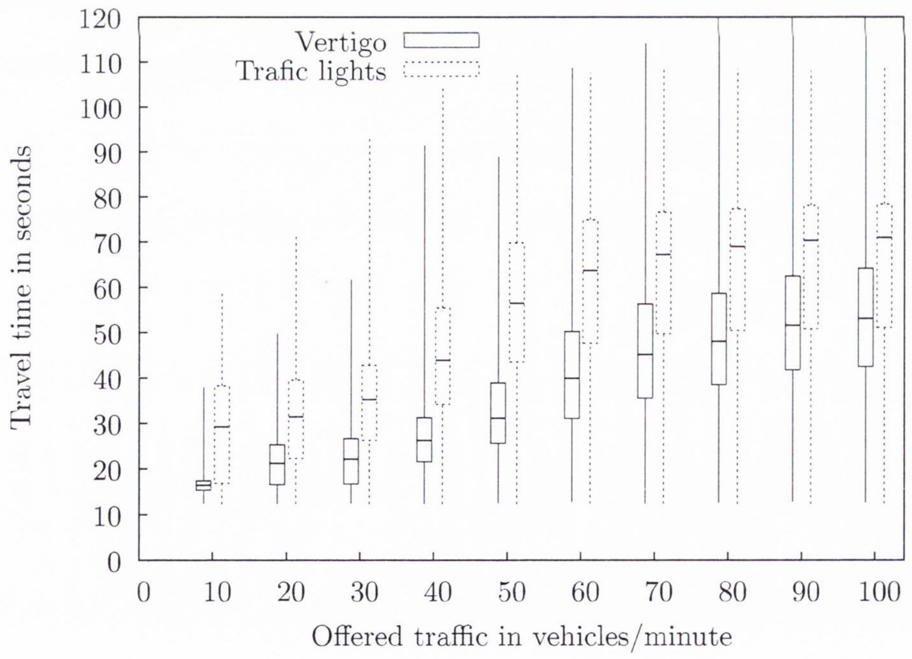


Fig. 6.15: Quartiles of the travel time over the intersection for different entry rates

at which the intersection reaches its maximum capacity. As shown in Figure 6.14, the intersection managed using Vertigo reaches a throughput of $\sim$ 37 vehicles per minute, which is a 25% increase from the throughput achieved using traffic lights of $\sim$ 30 vehicles per minute. This shows the potential of using Vertigo for cooperative automated driving to improve the efficiency of traffic. Due to the effects of queuing, travel times increase significantly for high entry rates as shown in Figure 6.15, but the median is still as much as 40% lower than the travel time for ordinary traffic lights.

The average success rate shown in Figure 6.12 remains relatively stable for higher entry rates, showing Vertigo can scale to a large number of vehicles. Overhead is significantly impacted by an increase in the number of vehicles as shown in Figure 6.13. The increase is more than linear, since query and beaconing traffic increase linearly, but traffic from responses increases quadratically. However, at the point where the entry and departure rates converge to $\sim$ 37 vehicles/minute, the overhead also stabilizes due to the fact that the number of vehicles being added to the scenario is reduced. Ultimately, the overhead is bounded by the maximum density of vehicles, based on their size and minimum distance, and the size of the target area of geocasts. Vertigo only needs to scale to a bounded number of vehicles.

6.3.4 WLAN Transmission power

The wireless network standard 802.11p [IEEE 10] defines 4 power classes:

- Class A: 1mW (0 dBm)
- Class B: 10mW (10 dBm)
- Class C: 100mW (20 dBm)
- Class D: 750mW (28.8 dBm)

It is up to regulators to decide under which circumstances each power class can be used, which is not yet clearly defined. The effects of using each of the power classes is

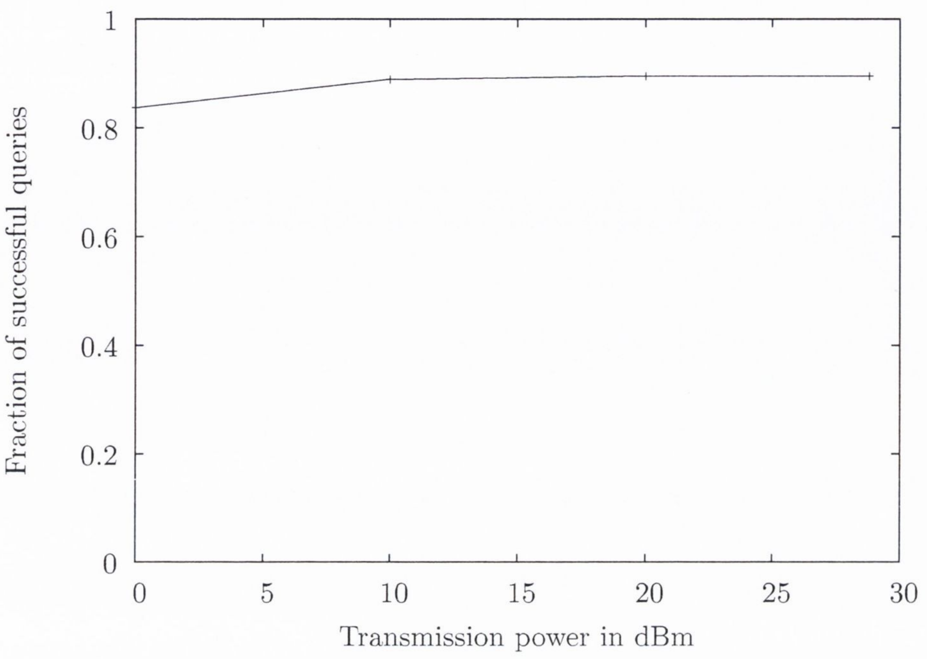


Fig. 6.16: The average success rate of queries for different transmission powers

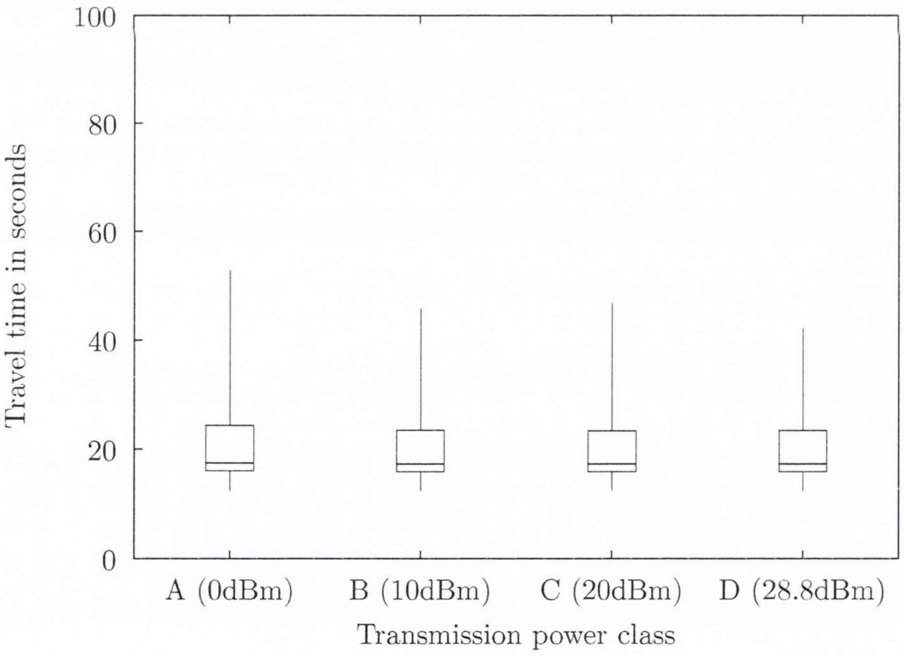


Fig. 6.17: Quartiles of the travel time for different transmission powers

studied in $4(classes) \times 5(seeds)$ simulation runs. For the purpose of evaluation, a greater transmission power means message transmissions can be successful over a greater range, but collisions also become possible over greater distances.

Figure 6.16 shows a slight drop in success rate for the lowest transmission power of 1mW. This drop has no significant effect on traffic rate or overhead, and the graphs for them are omitted. The only noticeable effect is a slight increase in travel time shown in Figure 6.17. When transmission power is low, there are slightly more cases in which the first query by a vehicle fails due to the range being too short to reach all vehicles in the delivery area. For transmission powers of 10dBm or above, the results are roughly the same in all cases. The recommended class for use of Vertigo for intersection management is therefore Class B, though Class A is sufficient.

6.4 Discussion

The Vertigo communication model offers rigorous safety guarantees that may come at the expense of the progress vehicles make in traffic. Vehicles should treat a geocast as having failed when insufficient information is available to confirm its success. This gives rise to the question whether and under what conditions it is feasible to implement the Vertigo model in such a way that it achieves success in a real scenario, and whether the achieved rate of success is sufficient to support a real application. The experiments presented in this Chapter show that in the intersection scenario, 70% or more of queries succeed, if the maximum range of LIDAR sensors is 30m or higher, the position inaccuracy bound is 2m or less, and the transmission power 1mW or higher. Even at success rates as low as 40%, the coordination protocol can sustain high rates of progress with a LIDAR range of 25m or higher, and a position inaccuracy bound of 3m or less, at the cost of increased network overhead. Vertigo can scale to dense traffic flows without substantial impact to success rates. It is helped by the fact that the number of vehicles that can participate in a query is bounded by physical constraints. When omission failures occur, channel overhead increases substantially, but not to the point where it saturates the channel.

Importantly, the coordination protocol using Vertigo outperforms traffic lights as an intersection management solution by increasing the traffic flow by $\sim 25\%$ and reducing expected and worst-case travel time by as much as 40%. These results are comparable to work by [Dresner 08] on centralized intersection management, while providing stronger safety guarantees, particularly robustness to omission failures. While improving traffic flow is not a particular goal of this thesis, these results legitimize the use of cooperative automated driving for that purpose, and more importantly, they show that the success rate achieved by Vertigo in this scenario is sufficiently high for building a cooperative automated driving system that can outperform an existing traffic management solution.

Chapter 7

Conclusions

As the closing part of this thesis, the main contributions and future work items are summarized.

7.1 Contributions

The primary contribution of this thesis is the Vertigo communication model. Existing communication solutions for cooperative automated driving systems are not robust to omission failures in broadcast or geocast, despite the strict safety requirements of automated driving. The Vertigo communication model specifies interfaces and data structures for geocast with reliable success confirmation in the presence of unbounded omission failures by combining acknowledgements with reliable, spatio-temporal membership. The characteristics of the model make it feasible to implement and applicable to the problem space.

Key to the Vertigo model is the use of membership tuples that define a relationship between the identifiers of vehicles, space, and time. Membership tuples can be decayed to derive information about the future from the past, merged to combine information from multiple sources and multiple points of time, generated using positioning and ranging sensors, and shared over a wireless network. Using membership tuples, a complete view of all vehicles that could be in a given area at a given time can be obtained. The Vertigo model defines a geocast operation for sending a message to all vehicles in a target area at a target time, to which those vehicles can respond. The sender of the geocast can collect responses and finally obtain the result of the geocast, which uses membership and the set of responders to confirm whether all vehicles in the target area at the target time received the message.

The feasibility of implementing the Vertigo model is demonstrated through an implementation using sensing capabilities that were already present in automated vehicles 8 years before this thesis was written [Buehler 07]. An important part of the implementation is a set of geometric algorithms that implement operations on areas relative to a road map. Algorithms are defined for containment, union, and shifting the boundaries of the area. These algorithms implement the operations on membership tuples required by the model, as well as operations for generating the delivery area of a geocast, and evaluating the success condition. To generate membership tuples, a transformation from LIDAR beams to a one-dimensional empty area is defined, taking position inaccuracy into consideration. Beaconing is used to share membership tuples in such a way that receivers can form a membership view for the area around them. Filtered broadcast and delayed unicast are used to implement geocast with responses. The feasibility of using the provided implementation to achieve successful queries in an actual coordination scenario is demonstrated through a series of simulations. The minimum required LIDAR range in the scenario to achieve maximal traffic rates was found to be 25m and position inaccuracy up to 3m can be tolerated. Even for high traffic densities and different transmission powers, overhead remains low enough not to saturate the channel and maintain a high rate of progress.

The applicability of the Vertigo model is demonstrated through its use in a coordination protocol for coordinating arbitrary intersections. The protocol is based on a model of tracks, interconnected line strings. Where tracks intersect, a conflict area exists. The protocol enables a vehicle to obtain an allocation for entering into a set of conflict areas defined by a (conflict) trajectory. The geocast operation from the Vertigo model is used to send an allocation request to a target area containing all vehicles who might have, or might be competing for, an allocation for the same conflict areas. If all vehicles in the target area accept or tentatively accept the allocation request, the sender may proceed into the conflict areas, provided that all vehicles that tentatively accepted the allocation have passed or released their allocation. An allocation is committed if the vehicles drives into a designated commit area within the time frame obtained in the allocation. The commit protocol is made robust to vehicles leaving an area without communication through the use of membership tuples. The applicability of the Vertigo model is further demonstrated by the fact that success rate achieved by the Vertigo implementation is high enough for the coordination protocol to outperform conventional traffic management by increasing the maximum rate of traffic over the junction by roughly 25%, and reducing the median travel time by up to 40%. These results are comparable to centralized intersection management approaches, but achieved in a way that is decentralized and robust to omission failures.

7.2 Future Work

The Vertigo communication model lays the foundation for a new class of coordination protocols for cooperative automated vehicle that, unlike previous protocols, can achieve safety in the presence of unbounded omission failures. New protocols are needed to support coordination scenarios such as roundabouts, overtaking, intersections with priority rules, lane merging, and platooning. A general-purpose protocol that supports each of these scenarios might also be feasible. Vertigo can potentially be used as the basis for protocols for traffic management of an entire city or highway network. Novel coordination models and protocols may be necessary to ensure and optimize progress across multiple intersections and roundabouts to avoid creating deadlocks. The roundasim simulator can be used to evaluate the protocols and Vertigo implementation in these large-scale scenarios, and find conditions under which success is achievable across a wider range of scenarios and parameters. The results found using roundasim in small-scale scenarios should be validated against real-world experiments.

Various improvements can be made to the Vertigo implementation, in particular to reduce reliance on sensor accuracy, and support three-dimensional sensing. The current implementation assumes vehicles drive in a perfect two-dimensional plane with regards to LIDAR sensors. In reality, vehicles may wobble, roads are not perfectly flat, and vehicles and obstacles may have irregular shapes. Distance measurements are dependent on the orientation of the vehicle in three-dimensional space and may be overestimated. To deal with this problem, more sensors and processing algorithms are needed to detect and correct for the slope of the road. Delay and noise in LIDAR measurements is currently not considered, nor is inaccuracy in the orientation of the vehicle and the sensors. These could be taken into account by shrinking the polygon representing the known empty area. The effects of clock inaccuracy and road map inaccuracy also need to be studied and possibly made explicit.

One of the biggest drawbacks of the current Vertigo implementation is the need for 100% deployment. The presence of a vehicle (or other traffic participant) that does not participate in the membership service or group communication protocol, is interpreted as a failure to reach all vehicles in the area by the sender of a geocast. There are several ways this problem might be dealt with. First of all, a coordination protocol could adapt to the area that is covered by membership tuples, even if incomplete. The coordination protocol in Chapter 5 is relatively simple, since it only considers success or failure of a query. A more sophisticated algorithm could consider partial success. Additionally, the implementation could take more physical restrictions than just the maximum speed into account, and try to bound the impact of incompatible vehicles. A vehicle in the membership view could act as a gatekeeper that temporarily prevents incompatible vehicles behind it from passing, to allow compatible vehicles to make progress. Incompatible vehicles could also be tracked with more precision. Potentially, the number of vehicles in an area can be known reliably, even if they are incompatible.

Many other solutions could be applied to deal with the partial deployment problem, which, to some extent, is the problem of automated driving itself. Vertigo could be used within a closed clusters of vehicles, either formed dynamically or enforced. Vehicles could coordinate their actions using Vertigo within the cluster, whereas conventional mechanisms are used when interacting with vehicles outside the cluster. A coordination protocol using Vertigo could also interact with traffic management infrastructure that controls regular traffic. Different slots of time or space could be allocated to compatible and incompatible vehicles. Given the relatively modest requirements of the Vertigo implementation in terms of sensing and communication capabilities compared to autonomous vehicles, Vertigo may one day be deployed as a software update in a world where most vehicles have sensors for ranging, positioning, and orientation, and ad-hoc networking.

Appendix A

API Notation

To specify APIs, an abstract notion of events is used, written in bold. Input events are generated by the application (control software), and trigger processing at and pass data to the service implementing the model. Output events are generated by the service and trigger processing at and pass data to the application. The definition also specifies the data types of parameters of the events in terms of tuples and sets. The notation composes sets and tuples of elements of a certain type, written with a capital letter, and optionally a unique name, written in subscript. For example, a tuple written as ($target \in$ Area, ...) means the first element of the tuple has type *Area* and name *target*. Below are some common types used to specify events.

Table A.1: API notation

type	Message = ($d \in$ Data,$s \in$ Size).
	Data = ($b_1 \in$ Byte, ..., $b_s \in$ Byte).
	Byte $\in \{n : n \in \mathbb{N} \wedge n < 256\}$.
	Size $\in \mathbb{N}$.
description	Binary messages. The size limitation is an artefact of the implementation.

type	Address = (Byte$_1$, Byte$_2$, ..., Byte$_{16}$).
description	The globally unique network-layer address of the vehicle.
type	Port $\in \{n : n \in \mathbb{N} \wedge 0 < n < 65536\}$.
description	The port identifies the endpoint on which the application is listening for a request or response message. At most one application can be listening on a port.
type	ID$_{domain} \in \mathbb{N}$.
description	An identifier that is unique within *domain*.
type	Time = $\mathbb{R}^+$.
description	Point in time. A greater value means further into the future.

Appendix B

Lower-layer APIs

The APIs defined in this appendix comprehensively specify the capabilities that the computational environment in which a Vertigo implementation operates needs to offer, apart from general purpose computing. In particular, APIs for ad-hoc networking, positioning, and radar are specified. The notation introduced in Appendix A is used to specify the interfaces.

B.0.1 Ad-hoc networking API

Vertigo requires an API for communication over an ad-hoc network. The API defines two transmission primitives: **unicast** and **broadcast**. Unicast uses a globally unique address to send a message to a single node, while broadcast sends a message to all nodes in the radio range of the sender. The **receive** primitive can be used to receive broadcasts and unicasts from other vehicles. The primitives can be implemented on top of an 802.11p [IEEE 10] network interface. The full specifications are as follows:

Table B.1: Communication API specifications

input	**broadcast**($m \in$ Message) at vehicle with address *source*.
description	The broadcast event triggers the communication service to send the data in m over a radio. The communication service of vehicles that successfully receive the message will trigger a **receive**($source, m$) event. There is no multi-hop forwarding involved.
input	**unicast**($m \in$ Message, $d \in$ Address) at vehicle with address *source*.
description	The semantics of **unicast** are identical to that of **broadcast**, except the **receive**($source, m$) event will only trigger at a vehicle that receives the message and has address d.
output	**receive**($source \in$ Address, $m \in$ Message) at vehicle with address d.
description	The receive event is triggered by the communication service at vehicles that successfully receive a message from another vehicle. It is not triggered at the vehicle that sent the message.

There are several communication techniques that may be very beneficial, but that are not currently considered in the design. In particular, the use of directed antennas [Gharavi 09] and transmission power control [Torrent-Moreno 06] are currently not considered. Future extensions of Vertigo may take advantage of these techniques, but they are not currently within the scope of the design.

B.0.2 Sensor APIs

Vertigo relies on the presence of several types of sensors. This section specifies the interfaces of the sensors that the Vertigo implementation relies on. The interfaces can be provided by devices that are commonly used in automated vehicles.

B.0.2.1 Position sensor

One of the most basic and important sensors available on a vehicle is a positioning sensor. Position can be provided by a satellite system like the Global Positioning System or Galileo, and can be augmented with inertial sensors, knowledge of driving actions, or relative positioning. For the sake of simplicity, the coordinates are assumed to be in a Cartesian system in which distance can be calculated using the Pythagorean theorem such as Universal Transverse Mercator [Karney 11]. The positioning sensor produces a two-dimensional point and a radius that indicates how far the actual position of the vehicle might lie from the point.

Table B.2: Position sensor specifications

output	**position**($p \in$ Point2D, $\alpha \in \mathbb{R}^+$).
description	The **position** event is triggered periodically and produces a two-dimensional point p indicating the approximate position of the vehicle and and an accuracy value α indicating the radius of a circle centred in p, in which the vehicle is guaranteed to be.
type	Point2D = ($x \in$ Easting, $y \in$ Northing). Easting, Northing $\in \mathbb{R}$.
description	Describes a two-dimensional point.

The accuracy value may vary depending on how the sensor is implemented and how much information it has available (e.g. the number of satellites from which a signal is received). A simple implementation would use a constant value. The guarantees provided by Vertigo are partially dependent on the validity of the accuracy value, which must thus be pessimistic. The impact of the inaccuracy bound is discussed in Chapter 6.

B.0.2.2 Orientation sensor

The orientation sensor is needed to interpret the results of the ranging sensors, which produce beams whose orientation needs to be determined using the orientation sensor.

Table B.3: Orientation sensor specifications

output	**orientation**($o \in$ Angle). Angle $= \{a : a \in \mathbb{R}, -\pi \leq a \leq \pi\}$.
description	The **orientation** event is periodically triggered and produces a value o indicating the approximate orientation of the vehicle in two dimensions where 0 is East and π is North.

A compass or repeated position measurements can be used to provide the orientation sensor interface. A GPS compass uses two receivers to provide highly accurate orientation within 1° [Tu 97]. Given that such a high accuracy can be achieved with basic off-the-shelf hardware, inaccuracy of the orientation sensor is not considered by the Vertigo implementation.

B.0.2.3 Ranging sensor

The Vertigo implementation uses a ranging sensor to measure areas in which there are no other vehicles. The output of the ranging sensor is represented as a set of beams at fixed angles that are relative to the orientation of the vehicle.

Table B.4: Ranging sensor specifications

output	**ranges**($s \in \mathrm{ID}_{sensors}, B \subset$ Beam).
description	The **ranges** event is periodically triggered and produces a set of beams B from a sensor with identifier s. The relative position, orientation, maximum range, angular resolution, and angular range of sensor s are known to applications.

type	Beam = $(d \in \mathbb{R}^+, a \in \text{Angle})$.
description	A beam represents a distance measurement of length d at relative angle a from the sensor. Inaccuracy of individual measurements is reflected by a reduction in distance d.

The ranging sensor can be provided by a LIDAR, such as those produced by SICK [SICK AG 13] and Velodyne [Velodyne 13]. Some processing may be required to translate an array of three-dimensional beams into a two-dimensional plane.

Appendix C

Road map specifications

The main components of Vertigo each make use of a road map and thus require a service that can be queried for a relevant section of the road map. The representation of the road map is assumed to be common among all vehicles, and known in advance. This appendix specifies the road map and area representations used in the Vertigo implementation.

C.1 Road segment graph

The road map used by Vertigo implementation is represented as a graph with geometric attributes. It provides a global frame of reference for geometric information, and defines restrictions which the Vertigo implementation uses to reason about the potential worst-case behaviour of unknown vehicles. The road map is divided into segments (named 'RoadSegment'). The segments represent edges in a graph, in which the nodes (named 'Connector') represent the possibility for vehicles to move from one segment to another. The individual segments and connectors have globally known identifiers that can be referred to in messages. This helps to avoid having to encode complex geometric information in messages. The formal specifications of the road map representation using the notation introduced in Appendix A are given below.

Table C.1: Road map specifications

output	**map**($S \subset$ RoadSegment, $C \subset$ Connector, $v \in \mathrm{ID}_{maps}$).
description	The road map service provides a set of road segments S and a set of connectors C, belonging to a road map with version number v.
type	RoadSegment = ($id \in \mathrm{ID}_{segments}$, $from \in \mathrm{ID}_{connectors}$, $to \in \mathrm{ID}_{connectors}$, $path \in$ Polyline, $surface \in$ Polyline, $direction \in$ Direction, $v_{max} \in \mathbb{R}^+$, $o_{max} \in$ Angle). Direction = $\{normal, both, reverse\}$. Angle = $\{a : a \in \mathbb{R}, -\pi \leq a \leq \pi\}$.
description	A road segment represents part of a road. Two road segments are connected if one of the connector identifiers, $from$ or to, of one segment is equal to one of the connector identifiers of the other road segment. While on the road segment, the speed of a vehicle may not be higher than v_{max} and its orientation may differ no more than o_{max} from the line segment(s) of *path*. The $surface$ polygon defines the two-dimensional shape of the road segment.

type	Connector = ($id \in \mathrm{ID}_{connectors}$, $segments \subset \mathrm{ID}_{segments}$).
description	A connector either represents a closed end of a road segment if $\lvert\mathrm{segments}\rvert = 1$ or a connection to one or more other road segments if $\lvert\mathrm{segments}\rvert > 1$. The existence of a connector for which segments contains road segment identifiers a and b is a pre-condition for vehicles being able to move between these two segments. However, other constraints apply which may restrict such movements (e.g. the segments may be uni-directional).
type	Polyline = $(p_1 \in \mathrm{Point2D}, ..., p_n \in \mathrm{Point2D})$.
description	Describes a sequence of points, representing connected line segments. The line segments of a single polyline may not intersect. If the polyline represents a polygon, the first and last point need to be identical.

As shown in the definition, road segments have a path and a surface associated with them. These are used to define and reason about geometric restrictions to the vehicle. Road segments also have attributes describing restrictions on speed, orientation, and direction. The restrictions are as follows:

- While on a road segment, the two-dimensional position of the vehicle is within the two-dimensional surface of the road segment.
- While on a road segment, the two-dimensional velocity of the vehicle may not exceed the v_{max} of the segment.
- While on a road segment, the one-dimensional velocity of the vehicle (or rather, the projection of its position on *path*) may not exceed the v_{max} of the segment.
- While on a road segment, the one-dimensional velocity of the vehicle may not exceed the v_{max} of the segment (defined in Section C.2).

- While on a road segment, the orientation of a vehicle may not differ more than o_{max} from the orientation of the line segment(s) of *path* on which the projection of the position of the vehicle lies.
- While on a road segment, a vehicle may only follow the *path* in the direction given by *direction*.
- A line drawn perpendicular to a line segment on *path* may only intersect *surface* once.

The primary purpose of the restrictions is to allow the membership service to compute decay. The validity of the constraints in the road map is critical to the guarantees offered by Vertigo. For example, if vehicles could exceed the maximum velocity, the membership service may falsely conclude that an area is void of unknown vehicles based on the maximum velocity constraint. As a result, Vertigo may falsely conclude that all vehicles in the delivery area received the message. On the other hand, a very conservative (high) maximum velocity would cause decay to occur more rapidly, and increase information gathering overhead and potentially a drop in success rate.

The restrictions are effectively a contract between the controllers of vehicles and Vertigo. As long as vehicles honour the contract, Vertigo can provide success confirmation reliably. To make such a contract feasible, restrictions can made very permissive. As vehicles become more automated, they can be made to follow stricter bounds, which would provide Vertigo with better upfront knowledge about the behaviour of vehicles, which may result in a better success rate and reduced overhead.

For simplicity, only two-dimensional geometry is considered in the road map representation. Since vehicles always stay on the surface, two-dimensional geometry is sufficient to enforce safety guarantees. Distance in terms of altitude does not need to be considered. The two-dimensional geometries of road segments are allowed to overlap to represent bridges and tunnels, provided that no connector exists between them.

C.2 Position specification

Vertigo mostly interacts with the part of the road map that represents the roads in the proximity of the vehicle. To be able to find this part in a practical manner, the position of the vehicle should be augmented with the identifier of the road segment the vehicle is currently on. The interface used by Vertigo for positioning provides augmented position both in a one-dimensional and two-dimensional form. The one-dimensional form is represented as an offset and the identifier of the road segment that the vehicle is on. The two-dimensional form is represented as a two-dimensional point and the identifier of the road segment that the vehicle is on. The type definitions are given below.

Table C.2: Position specifications

type	Position1D $= (offset \in \mathbb{R}^+, segment \in \mathrm{ID}_{segments})$.
description	Describes the position of a vehicle as a point obtained by walking a distance given by offset along the path of road segment *segment* in the order of the points. The offset may not be greater than the total length of the path.
type	Position2D $= (point \in \mathrm{Point2D}, segment \in \mathrm{ID}_{segments})$.
description	Describes the position of a vehicle as a two-dimensional point *point* on a road segment with identifier s.

To simplify algorithms, and allow compact messages, two-dimensional geometry can be converted to one-dimensional geometry. A one-dimensional position is represented as an *offset* on the path of a road segment. One-dimensional speed is represented is the increase or decrease in offset per time unit. A one-dimensional position can be converted into a two-dimensional position by walking a distance of *offset* along the path from the start. A two-dimensional position can be converted to a one-dimensional position by finding the offset of the closest point on the path.

Which road segment the vehicle is on is assumed to be known reliably at the start. In a real setting, a method for transitioning to autonomous mode [Prada Gomez 11] could be used for this purpose. From that point forward, the positioning system periodically checks whether the two-dimensional position of the vehicle with respect to the surface of the road segment the vehicle is on and other road segments that share a common connector. The position of a vehicle needs to stay within the surface polygon of a road segment, and may not move from one road segment to the other unless the road segments share a connector and their surface polygons share at least one surface segment.

C.3 Area specification

Vertigo uses two different one-dimensional representations of areas that are relative to the paths of road segments in the road map. Both representations specify a section of the road map, but one is optimized for compactness in message, and the other for reasoning and transformations. This section specifies both representations and define the translations between them. The two representations of an area provide a framework for geographically-defined communication. The area representations are used for defining the target area of a geocast, the delivery area in geocast messages, and the membership area in membership tuples.

C.3.1 Boundary-based representation

In messages, areas are represented as a part of the road map that is sealed off by a set of one-dimensional boundaries on the paths of road segments. The number of boundaries needed to represent an area depends on the size of the area and the complexity of the road map. For example, an area covering a long stretch of highway can be represented by boundaries at a starting and ending offset on the highway, and boundaries for all exits and entries in between. An area covering a simple four-way intersection can be represented by 4 boundaries. The benefit of the boundary representation, apart from compactness, is that the borders of the area are explicit, which is used to apply transformations to shrink and expand the area from the borders. The boundary representation is defined as follows:

Table C.3: Area representation for messages

type	BoundaryArea1D $\subset$ Boundary1D.
description	In messages, an area is defined by a set of one-dimensional boundaries. The boundaries must seal off part of the road network. If not, the area is invalid.
type	Boundary1D = ($segment \in \text{ID}_{segments}$, $offset \in \mathbb{R}^+$, $inclusive \in$ Inclusive1D). Inclusive1D $= \{front, back\}$.
description	A one-dimensional boundary is defined by a distance $offset$ along the path of road segment $segment$. $inclusive$ indicates whether the part of the road behind or in front of the boundary is included in the area, given the direction of the *path* of the road segment.

The boundary representation is used to compactly represent areas in messages. However, it is not a practical representation for testing presence or transformations on the area. Given a set of boundaries, it is not obvious whether a path offset on a given road segment is included in the delivery area or not. To be able to perform such reasoning in a practical manner, the boundaries can be translated to a set of non-overlapping road segment ranges.

C.3.2 Range-based representation

To implement merging and containment operations efficiently, an area can be represented as a set of one-dimensional ranges of road segments. The ranges are defined by a *start* and an *end* offset on the path of a road segment, with $end > start$. Tests and transformations on the area can be broken down into tests and operations on ranges. For example, testing whether a point with path offset o on road segment i is included in the area can be done by looking up the set of ranges R_i for road segment i and evaluating whether $\exists r \in R_i : r.start \leq o \leq r.end$ holds true. The ranges are indexed by road segment identifier, such that the expected look up time for a set of ranges on a road segment is constant. The full specifications of the range-based area representation are as follows:

Table C.4: Internal area representation

type	RangeArea1D $\subset$ RoadSegmentRange1D.
description	A range-based area is defined as a set of non-overlapping road segment ranges. Points that fall within any of the ranges are included in the area. For efficiency, the ranges can by indexed by road segment identifier.
type	RoadSegmentRange1D = ($start \in \mathbb{R}^+$, $end \in \mathbb{R}^+$, $segment \in \mathrm{ID}_{segments}$) with $start < end$.
description	A road segment range is defined as the *start* offset and the *end* offset on the path of road segment *segment*. Points in between start and end are included in the range.

C.3.3 Boundaries to ranges translation

When a Vertigo receives a message containing a boundary representation of an area, it first translates it into the range-based representation using the **boundariesToRanges** function. To do so, it performs a breadth-first search (BFS) on the road map from a selected starting boundary to the other boundaries with which it forms a contiguous area. This process is repeated until all boundaries were either selected as a starting point or found in the search.

The search starts in the inclusive direction of the starting boundary. If another boundary on the same road segment exists in the opposite direction, the range between the boundaries is added to the set of ranges and the search is done. If no such boundary exist, a range between the boundary and the other end of the path is included in the set and the search continues from the connector. From a connector, each of the connected road segments are traversed. If no boundary exists on a road segment, a range is added to the set with start $= 0$, end $=$ **length**($path$), and the connector on the other end of the road segment is included in the BFS queue. If a boundary does exist, a range from the connector to the boundary is added to the set and the search continues to traverse other road segments. If all road segments are traversed and the queue is empty, the search is done. For completeness, the full algorithm is given below. When the algorithm terminates, result contains the set of ranges following the previously defined format.

```
func boundariesToRanges($A_b \in$ BoundaryArea1D).
$A_r \leftarrow \emptyset$
unusedBoundaries $\leftarrow \emptyset$
groupedBoundaries $\leftarrow \emptyset$
for all boundary $\in A_b$ do
    groupedBoundaries[boundary.segment] $\leftarrow$ groupedBoundaries[boundary.segment] $\cup$
    {boundary}.
end for
while |unusedBoundaries| $> 0$ do
    rootBoundary $\leftarrow$ removeAny(unusedBoundaries).
    segment $\leftarrow$ segments[rootBoundary.segment].
    boundaryGroup $\leftarrow$ groupedBoundaries[rootBoundary.segment].
    closestBoundary $\leftarrow$ findNextBoundary(rootBoundary, boundaryGroup).
    if closestBoundary then
    $A_r \leftarrow A_r \cup \{($**min**$($rootBoundary.offset, closestBoundary.offset$),$
                    **max**$($rootBoundary.offset, closestBoundary.offset$),$
                    rootBoundary.segment$)\}$.
    unusedBoundaries $\leftarrow$ unusedBoundaries $\setminus$ $\{$closestBoundary$\}$.
  else
    bfsQueue $\leftarrow$ [].
    visitedConnectors $\leftarrow \emptyset$.
    if rootBoundary.inclusive = front then
      $A_r \leftarrow A_r \cup \{($rootBoundary.offset,
                    **length**(segment.path),
                    rootBoundary.segment$)\}$.
      **push**(bfsQueue, segment.to).
      visitedConnectors $\leftarrow$ visitedConnectors $\cup$ $\{$segment.to$\}$.
    else
      $A_r \leftarrow A_r \cup \{(0,$ rootBoundary.offset, rootBoundary.segment$)\}$.
      **push**(bfsQueue, segment.from).
      visitedConnectors $\leftarrow$ visitedConnectors $\cup$ $\{$segment.from$\}$.
    end if
    while $|$bfsQueue$| > 0$ do
      connectorID $\leftarrow$ **pop**(bfsQueue).
      connector $\leftarrow$ connectors[connectorID].
      for all segmentID $\in$ connector.segments do
        segment $\leftarrow$ segments[segmentID].
        if connectorID = segment.from then
          otherConnectorID $\leftarrow$ segment.to.
          arrivalOffset $\leftarrow$ 0.
        else
          otherConnectorID $\leftarrow$ segment.from.
```

```
                arrivalOffset ← **length**(segment).
            **end if**
            **if** otherConnectorID $\notin$ visitedConnectors **then**
                boundaryGroup ← groupedBoundaries[rootBoundary.segment].
                closestBoundary ← **findNextBoundary**(rootBoundary, boundaryGroup).

                **if** closestBoundary **then**
                    $A_r \leftarrow A_r \cup \{($**min**(arrivalOffset, closestBoundary.offset),
                                 **max**(arrivalOffset, closestBoundary.offset),
                                 segmentID$)\}$.
                    unusedBoundaries ← unusedBoundaries $\setminus$ {closestBoundary}.
                **else**
                    $A_r \leftarrow A_r \cup \{(0,$ **length**(segment), segmentID$)\}$.
                    **push**(bfsQueue, otherConnectorID).
                    visitedConnectors ← visitedConnectors $\cup$ {otherConnectorID}.
                **end if**
            **end if**
        **end for**
    **end while**
  **end if**
**end while**
**return** $A_r$
**end boundariesToRanges**.
```

A weakness of the translation using BFS is that it is possible for a malicious node to specify a set of boundaries that does not fully seal off an area. In this case, the BFS algorithm might not terminate until it has traversed the entire road network. A practical solution to this problem is to limit the diameter of the delivery area graph. The algorithm can keep track of the number of vertices it traversed to reach a given node. For the scope of this thesis, non-malicious implementations are assumed.

C.3.4 Ranges to boundaries translation

When a message containing an area is constructed, the range-based representation needs to be translated into the boundary-based representation using the **rangesToBoundaries** function. In this case, the main problem is to determine whether or not the start or end of a range is a boundary. For a given range $(start, end, id)$, if $start > 0$ then a boundary $(id, start, front)$ is added to the set. If $end < \mathbf{length}(path)$ for the path of the road segment with identifier id, then a boundary $(id, end, back)$ is added to the set. In the alternative case, a range touches the *from* connector of a road segment if start $= 0$, and a range touches the *to* connector of a road segment if end $\geq$ **length**(path). If a range touches a connector, the algorithm determines if each connecting road segment has a range that touches the same connector using the **isConnectorCovered** function. If not, a boundary is placed at the connector - $(id, 0.0, front)$ for connector *from*, $(id, \mathbf{length}(path), back)$ for connector *to*. The full algorithm is given below.

```
func rangesToBoundaries($A_r \in$ RangeArea1D).
$A_b \leftarrow \{\}$.
for all $r \in A_r$ do
  segment $\leftarrow$ segments[r.segment].
  if $r$.start $> 0 || \neg$isConnectorCovered(segment.from) then
    $A_b \leftarrow A_b \cup \{(\text{segment.id}, r.\text{start}, \text{front})\}$.
  end if
  if $r$.end $<$ length(segment.path)$|| \neg$isConnectorCovered(segment.to) then
    $A_b \leftarrow A_b \cup \{(\text{segment.id}, r.\text{end}, \text{back})\}$.
  end if
end for
return $A_b$.
end rangesToBoundaries.
```

Bibliography

[Allal 12] S. Allal & S. Boudjit. *Geocast Routing Protocols for VANETs: Survey and Guidelines.* In International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), pages 323–328, 2012.

[Asplund 12] Mikael Asplund, Atif Manzoor, Melanie Bouroche, Siobhan Clarke & Vinny Cahill. *A Formal Approach to Autonomous Vehicle Coordination.* In International Symposium on Formal Methods (FM), pages 52–67, 2012.

[Bachir 03] A. Bachir & A. Benslimane. *A multicast protocol in ad hoc networks inter-vehicle geocast.* In IEEE Vehicular Technology Conference Spring, volume 4, pages 2456–2460, 2003.

[Bai 07] Fan Bai & Hariharan Krishnan. *Reliability Analysis of DSRC Wireless Communication for Vehicle Safety Applications.* Rapport technique, General Motors Corporation, 2007.

[Barr 05] Rimon Barr, Zygmunt J. Haas & Robbert van Renesse. Scalable Wireless Ad hoc Network Simulation, pages 297–311. CRC Press, August 2005.

[Behrisch 11] M. Behrisch, L. Bieker, J. Erdmann & D. Krajzewicz. *SUMO*

- *Simulation of Urban MObility: An Overview.* In International Conference on Advances in System Simulation (SIMUL), Barcelona, Spain, 2011.

[Board 10] Transportation Research Board. Highway capacity manual. Special report. National Research Council (U.S.)., 2010.

[Bouroche 06a] M. Bouroche, B. Hughes & V. Cahill. *Building reliable mobile applications with space-elastic adaptation.* In International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), pages 627–632, 2006.

[Bouroche 06b] M. Bouroche, B. Hughes & V. Cahill. *Real-time coordination of autonomous vehicles.* In Intelligent Transportation Systems Conference (ITSC), pages 1232–1239. IEEE, 2006.

[Bu 10] Fanping Bu, Han-Shue Tan & Jihua Huang. *Design and field testing of a Cooperative Adaptive Cruise Control system.* In American Control Conference (ACC), 2010, pages 4616–4621, 2010.

[Buehler 07] Martin Buehler, Karl Iagnemma & Sanjiv Singh. The 2005 DARPA Grand Challenge: The Great Robot Race. Springer Publishing Company, Incorporated, 1st edition, 2007.

[Buehler 09] M. Buehler, K. Iagnemma & S. Singh. The DARPA Urban Challenge: Autonomous Vehicles in City Traffic. Springer Tracts in Advanced Robotics. Springer, 2009.

[Cardei 02] Mihaela Cardei, Xiaoyan Cheng, Xiuzhen Cheng & Ding zhu Du. *Connected Domination in Multihop Ad Hoc Wireless Net-*

works. In International Conference on Computer Science and Informatics (CS&I), 2002.

[Chan 11] Eric Chan, Peter Gilhead, Pavel Jelinek & Petr Krej. *SARTRE cooperative control of fully automated platoon vehicles.* In 18th World Congress on Intelligent Transport Systems, October 2011.

[Chang 99] Xinjie Chang. *Network simulations with OPNET.* In Simulation Conference Proceedings, Winter, volume 1, pages 307–314, 1999.

[Chen 06] Qi Chen, Daniel Jiang, Vikas Taliwal & Luca Delgrossi. *IEEE 802.11 based vehicular communication simulation design for NS-2.* In International workshop on Vehicular ad hoc networks (VANET), pages 50–56. ACM Press, 2006.

[Chockler 01] Gregory V. Chockler, Idit Keidar & Roman Vitenberg. *Group communication specifications: a comprehensive study.* ACM Computer Surveys, vol. 33, no. 4, pages 427–469, December 2001.

[Clemmons 58] Larry Clemmons, Chuck Downs & John W. Dunn. *Magic Highway U.S.A.* Disneyland (TV), May 1958.

[Corbett 12] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal

Szymaniak, Christopher Taylor, Ruth Wang & Dale Woodford. *Spanner: Google's globally-distributed database.* In USENIX conference on Operating Systems Design and Implementation (OSDI), pages 251–264. USENIX Association, 2012.

[de Nooij 10] Margriet van Schijndel de Nooij, Bastiaan Krosse, Thijs van den Broek, Sander Maas, Ellen van Nunen, Han Zwijnenberg, Anna Schieben, Henning Mosebach, Nick Ford, Mike McDonald, David Jeffery, Jinan Piao & Javier Sanchez. *Definition of necessary vehicle and infrastructure systems for Automated Driving.* Study Report SMART 2010/0064, European Commission, DG Information Society and Media B-1049 BRUSSELS Belgium, 2010.

[Demmel 12] S. Demmel, A. Lambert, D. Gruyer, A. Rakotonirainy & E. Monacelli. *Empirical IEEE 802.11p performance evaluation on test tracks.* In Intelligent Vehicles Symposium (IV), pages 837–842. IEEE, 2012.

[Dresner 08] Kurt Dresner & Peter Stone. *A Multiagent Approach to Autonomous Intersection Management.* Journal of Artificial Intelligence Research, vol. 31, pages 591–656, 2008.

[Elkaim 08] G.H. Elkaim, M. Lizarraga & L. Pedersen. *Comparison of low-cost GPS/INS sensors for Autonomous Vehicle applications.* In Position, Location and Navigation Symposium, pages 1133–1144. IEEE/ION, 2008.

[Fellendorf 01] M. Fellendorf & P. Vortisch. *Validation of the Microscopic Traffic Flow Model VISSIM in Different Real-World Situations.* Rapport technique, PTV AG, 2001.

[Gharavi 09] H. Gharavi & Bin Hu. *Directional Antenna for Multipath Ad Hoc Routing.* In Consumer Communications and Networking Conference (CCNC), pages 1 –5. IEEE, 2009.

[Hadim 06] S. Hadim & N. Mohamed. *Middleware: middleware challenges and approaches for wireless sensor networks.* Distributed Systems Online, IEEE, vol. 7, no. 3, pages 1–1, 2006.

[Hendriks 10] F. Hendriks, M. Tideman, R. Pelders, R. Bours & X. Liu. *Development tools for active safety systems: Prescan and VeHIL.* In International Conference on Vehicular Electronics and Safety (ICVES), pages 54–58. IEEE, 2010.

[Hermann 07] S.D. Hermann, C. Michl & A. Wolisz. *Time-Stable Geocast in Intermittently Connected IEEE 802.11 MANETs.* In Vehicular Technology Conference, Fall, pages 1922 –1926, 2007.

[Horowitz 04] Roberto Horowitz, Chin-Woo Tan & Xiaotian Sun. *An Efficient Lane Change Maneuver for Platoons of Vehicles in an Automated Highway System.* Rapport technique, Institute of Transportation Studies, UC Berkeley, 2004.

[Huang 06] Jihua Huang & H.-S. Tan. *A Low-Order DGPS-Based Vehicle Positioning System Under Urban Environment.* IEEE/ASME Transactions on Mechatronics, vol. 11, no. 5, pages 567 –575, 2006.

[Hughes 06] Barbara Hughes. *Hard Real-Time Communication for Mobile Ad Hoc Networks.* PhD thesis, Trinity College Dublin, October 2006.

[Ibrahim 09] K. Ibrahim, M. C. Weigle & M. Abuelela. *p-IVG: Probabilistic*

Inter-Vehicle Geocast for Dense Vehicular Networks. In Vehicular Technology Conference, Spring, pages 1–5. IEEE, 2009.

[IEEE 10] IEEE. *IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments.* IEEE Std 802.11p-2010, pages 1 –51, 2010.

[IRE 52] IRE. *Transactions of the IRE Professional Group on Vehicular Communications.* vol. 1, no. 1, 1952.

[Jackson 07] J. Jackson. *Microsoft robotics studio: A technical introduction.* Robotics Automation Magazine, IEEE, vol. 14, no. 4, pages 82–87, 2007.

[Jaynes 13] Nick Jaynes. *Smarter, safer, self-driving: 4 (almost) autonomous cars you can own today.* http://www.digitaltrends.com/cars/autonomy-today-fewer-crashes-tomorrow-five-current-cars-with-autonomous-tech/, January 2013.

[Jochle 12] T. Jochle, B. Wiedersheim, F. Schaub & M. Weber. *Efficiency analysis of geocast target region specifications for VANET applications.* In Vehicular Networking Conference (VNC), pages 250–257. IEEE, 2012.

[Karagiannis 11] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin & T. Weil. *Vehicular Networking: A Survey and Tutorial on Requirements, Architectures, Challenges, Standards and*

Solutions. Communications Surveys Tutorials, IEEE, vol. 13, no. 4, pages 584–616, 2011.

[Karney 11] C. F. F. Karney. *Transverse Mercator with an accuracy of a few nanometers.* Journal of Geodesy, vol. 85, pages 475–485, August 2011.

[Kim 98] Deok-Soo Kim. *Polygon offsetting using a Voronoi diagram and two stacks.* Computer-Aided Design, vol. 30, no. 14, pages 1069 – 1076, 1998.

[Kim 99] K.H. Kim. *Group communication in real-time computing systems: issues and directions.* In Workshop on Future Trends of Distributed Computing Systems, pages 252 –258. IEEE, 1999.

[Ko 98] Young-Bae Ko & Nitin H. Vaidya. *Location-aided routing (LAR) in mobile ad hoc networks.* In Annual ACM/IEEE international conference on Mobile computing and networking (MobiCom), pages 66–75. ACM, 1998.

[Ko 99] Y.-B. Ko & N.H. Vaidya. *Geocasting in mobile ad hoc networks: location-based multicast algorithms.* In Workshop on Mobile Computing Systems and Applications (WMCSA), pages 101 –110. IEEE, 1999.

[Le Lann 11] G. Le Lann. *Cohorts and groups for safe and efficient autonomous driving on highways.* In Vehicular Networking Conference (VNC), pages 1–8. IEEE, 2011.

[Le Lann 12] Gérard Le Lann. *On the Power of Cohorts - Multipoint Protocols for Fast and Reliable Safety-Critical Communications in Intelligent Vehicular Networks.* In International Conference on

Connected Vehicles and Expo (ICCVE). IEEE, TRB, ACM, IFAC, 2012.

[Lee 12] Joyoung Lee & Byungkyu Park. *Development and Evaluation of a Cooperative Vehicle Intersection Control Algorithm Under the Connected Vehicles Environment.* Transactions on Intelligent Transportation Systems, IEEE, vol. 13, no. 1, pages 81–90, 2012.

[Levinson 11] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J.Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling & S. Thrun. *Towards fully autonomous driving: Systems and algorithms.* In Intelligent Vehicles Symposium (IV), pages 163–168. IEEE, 2011.

[Lidstrom 12] K. Lidstrom, K. Sjoberg, U. Holmberg, J. Andersson, F. Bergh, M. Bjade & S. Mak. *A Modular CACC System Integration and Design.* IEEE Transactions on Intelligent Transportation Systems, vol. 13, no. 3, pages 1050–1061, 2012.

[Luettel 12] T. Luettel, M. Himmelsbach & H. J Wuensche. *Autonomous Ground Vehicles - Concepts and a Path to the Future.* Proceedings of the IEEE, vol. 100, no. Special Centennial Issue, pages 1831–1839, 2012.

[Luo 04] Jun Luo, Patrick Th. Eugster & Jean-Pierre Hubaux. *Probabilistic Reliable Multicast in Ad Hoc Networks.* Elsevier Ad Hoc Networks, vol. 2, no. 4, pages 369 – 386, 2004.

[Lynch 96] Nancy Ann Lynch. Distributed algorithms. Data Management Series. Morgan Kaufmann Publishers, 1996.

[Maihofer 03] C. Maihofer, C. Cseh, W. Franz & R. Eberhardt. *Performance evaluation of stored geocast.* In Vehicular Technology Conference (VTC), Fall, volume 5, pages 2901 – 2905, 2003.

[Maxemchuk 07] N. F. Maxemchuk, P. Tientrakool & T. L. Willke. *Reliable Neighborcast.* IEEE Transactions on Vehicular Technology, vol. 56, no. 6, pages 3278–3288, nov. 2007.

[Milanes 11] V. Milanes, J. Alonso, L. Bouraoui & J. Ploeg. *Cooperative Maneuvering in Close Environments Among Cybercars and Dual-Mode Cars.* IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 1, pages 15–24, 2011.

[Moras 10] J. Moras, V. Cherfaoui & P. Bonnifait. *A lidar perception scheme for intelligent vehicle navigation.* In International Conference on Control Automation Robotics Vision (ICARCV), pages 1809–1814, 2010.

[Navas 97] Julio C. Navas & Tomasz Imielinski. *GeoCast—geographic addressing and routing.* In Annual ACM/IEEE international conference on Mobile computing and networking (MobiCom), pages 66–76. ACM, 1997.

[Nett 03] E. Nett & S. Schemmer. *Reliable real-time communication in cooperative mobile applications.* IEEE Transactions on Computers, vol. 52, no. 2, pages 166 – 180, feb. 2003.

[O'Hara 12] Niall O'Hara, Marco Slot, Dan Marinescu, Jan Čurn, Dawei Yang, Mikael Asplund, Mélanie Bouroche, Siobhán Clarke & Vinny Cahill. *MDDSVsim: an integrated traffic simulation platform for autonomous vehicle research.* In International workshop on Vehicular Traffic Management for Smart Cities (VTM). IEEE Computer Society, 2012.

[Panichpapiboon 12] S. Panichpapiboon & W. Pattara-Atikom. *A Review of Information Dissemination Protocols for Vehicular Ad Hoc Networks.* IEEE Communications Surveys Tutorials, vol. 14, no. 3, pages 784–798, 2012.

[Ploeg 11] J. Ploeg, B.T.M. Scheepers, E. van Nunen, N. Van de Wouw & H. Nijmeijer. *Design and experimental evaluation of cooperative adaptive cruise control.* In International Conference on Intelligent Transportation Systems (ITSC), pages 260–265. IEEE, 2011.

[Ploeg 12] J. Ploeg, S. Shladover, H. Nijmeijer & N. van de Wouw. *Introduction to the Special Issue on the 2011 Grand Cooperative Driving Challenge.* IEEE Transactions on Intelligent Transportation Systems, vol. 13, no. 3, pages 989–993, 2012.

[Prada Gomez 11] Luis Ricardo Prada Gomez, Nathaniel Farfield, Andy Szybalski, Philip Nemec & Christopher Urmson. *Transitioning a mixed-mode vehicle to autonomous mode.* Patent, 12 2011. US 8078349.

[Proakis 08] J.G. Proakis & M. Salehi. Digital communications. McGraw-Hilll higher education. McGraw-Hill Higher Education, 2008.

[Roman 01] G.-C. Roman, Qingfeng Huang & A. Hazemi. *Consistent group membership in ad hoc networks.* In International Conference on Software Engineering (ICSE), pages 381 – 388, 2001.

[Ros 09] F.J. Ros, P.M. Ruiz & I. Stojmenovic. *Reliable and Efficient*

Broadcasting in Vehicular Ad Hoc Networks. In Vehicular Technology Conference, Spring, pages 1 –5. IEEE, 2009.

[Shladover 08] S.E. Shladover. *AHS research at the California PATH program and future AHS research needs.* In International Conference on Vehicular Electronics and Safety (ICVES), pages 4–5. IEEE, 2008.

[SICK AG 13] SICK AG. *SICK Group — Sensor Intelligence.* `http://www.sick.com/`, April 2013.

[Sin 11] Mong Leng Sin, M. Bouroche & V. Cahill. *Scheduling of Dynamic Participants in Real-Time Distributed Systems.* In Symposium on Reliable Distributed Systems (SRDS), pages 245–254. IEEE, 2011.

[Slot 11a] M. Slot & V. Cahill. *End-to-end acknowledgement of geocast in vehicular networks.* In Vehicular Networking Conference (VNC), pages 131–138. IEEE, 2011.

[Slot 11b] M. Slot & V. Cahill. *A Reliable Membership Service for Vehicular Safety Applications.* In Intelligent Vehicles Symposium. IEEE, 2011.

[Sobeih 04] A. Sobeih, H. Baraka & A. Fahmy. *ReMHoc: a reliable multicast protocol for wireless mobile multihop ad hoc networks.* In Consumer Communications and Networking Conference, pages 146–151. IEEE, 2004.

[Steen 12] Maarten Steen, Guillaume Pierre & Spyros Voulgaris. *Challenges in very large distributed systems.* Journal of Internet Services and Applications, vol. 3, no. 1, pages 59–66, 2012.

[Taysi 12] Z.C. Taysi & A.G. Yavuz. *Routing Protocols for GeoNet: A Survey.* IEEE Transactions on Intelligent Transportation Systems, vol. 13, no. 2, pages 939–954, 2012.

[Tonguz 06] O. K. Tonguz, N. Wisitpongphan, J. S. Parikh, Fan Bai, P. Mudalige & V. K. Sadekar. *On the Broadcast Storm Problem in Ad hoc Wireless Networks.* In International Conference on Broadband Communications, Networks and Systems (BROADNETS), pages 1–11, 2006.

[Tonguz 10] O.K. Tonguz, N. Wisitpongphan & Fan Bai. *DV-CAST: A distributed vehicular broadcast protocol for vehicular ad hoc networks.* IEEE Wireless Communications, vol. 17, no. 2, pages 47 –57, 2010.

[Torrent-Moreno 06] M. Torrent-Moreno, P. Santi & H. Hartenstein. *Distributed Fair Transmit Power Adjustment for Vehicular Ad Hoc Networks.* In Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON), volume 2, pages 479 –488, 2006.

[Treiber 13] Martin Treiber & Arne Kesting. Traffic flow dynamics. Springer, 2013.

[Trivedi 11] H. Trivedi, P. Veeraraghavan, S. Loke, A. Desai & J. Singh. *Routing mechanisms and cross-layer design for Vehicular Ad Hoc Networks: A survey.* In Symposium on Computers Informatics (ISCI), pages 243–248. IEEE, 2011.

[Tsugawa 02] S. Tsugawa. *Inter-vehicle communications and their applications to intelligent vehicles: an overview.* In Intelligent Vehicle Symposium (IV), volume 2, pages 564–569. IEEE, 2002.

[Tsukada 10] Manabu Tsukada, Ines Ben Jemaa, Menouar Hamid, Zhang Wenhui, Goleva Maria & Thierry Ernst. *Experimental Evaluation for IPv6 over VANET Geographic routing.* In International Wireless Communications and Mobile Computing Conference (IWCMC). IEEE, 2010.

[Tu 97] C.-H. Tu, K.Y. Tu, F.-R. Chang & L.-S. Wang. *GPS compass: novel navigation equipment.* IEEE Transactions on Aerospace and Electronic Systems, vol. 33, no. 3, pages 1063–1068, 1997.

[Urmson 08] C. Urmson & W. Whittaker. *Self-Driving Cars and the Urban Challenge.* IEEE Intelligent Systems, vol. 23, no. 2, pages 66–68, 2008.

[Van den Broek 10] T. H A Van den Broek, B.D. Netten, M. Hoedemaeker & J. Ploeg. *The experimental setup of a large field operational test for cooperative driving vehicles at the A270.* In International Conference on Intelligent Transportation Systems (ITSC), pages 198–203. IEEE, 2010.

[Velodyne 13] Velodyne. *Velodyne Lidar.* `http://velodynelidar.com/`, April 2013.

[Vivid Solutions Inc. 13] Vivid Solutions Inc. *Java Topology Suite.* `http://www.vividsolutions.com/jts/`, April 2013.

[Willke 05] Theodore L. Willke & Nicholas F. Maxemchuk. *Coordinated Interaction Using Reliable Broadcast in Mobile Wireless Networks.* In NETWORKING, pages 1168–1179. 2005.

[Willke 09] T. L. Willke, P. Tientrakool & N. F. Maxemchuk. *A survey of inter-vehicle communication protocols and their applications.*

IEEE Communications Surveys & Tutorials, vol. 11, no. 2, pages 3–20, jun. 2009.

[Yu 08] Qiangyuan Yu & G. Heijenk. *Abiding Geocast for Warning Message Dissemination in Vehicular Ad Hoc Networks.* In International Conference on Communications Workshops (ICC), pages 400–404. IEEE, 2008.