UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA


# Design and implementation of a protocol for safe cooperation of self-driving cars

João Pedro Vicente e Campos Pinto

**Mestrado em Engenharia Informática**
Especialização em Arquitetura, Sistemas e Redes de Computadores

Dissertação orientada por:
Prof. Doutor António Casimiro Ferreira da Costa
e co-orientada pelo Prof. Doutor Naercio David Pedro Magaia

2019

# Acknowledgments

My advisor's António Casimiro and Naercio Magaia, for their guidance during the course of this dissertation and Tiago for all the help and availability to discuss ideas regarding both our projects.

Cheila, for listening and encouraging me for the past nine months and always being by my side for the past five years.

My Mother, my Father, João Paulo and the rest of my family, for always supporting and encouraging me to go further.

Francisco, João Becho, João Batista, Nuno Rodrigues and others at LASIGE for the comradery and helpful debates.

Tomé, Emanuel, Geraldes, Pedro, Luís, Sofia, Bruno, Caça, Guilherme and everyone else at BARCODEU for the fun moments during the more stressful times.

*In the beginning the Universe was created. This has made a lot of people very angry and been widely regarded as a bad move.*

# Resumo

Desde que começou a ser produzido no início do século XX, o automóvel alterou profundamente a forma como as pessoas se deslocam e as cidades são construídas. À medida que a ecónomia dos paises se fortaleceu, também a popularidade do automóvel aumentou, e hoje em dia é extremamento comum existir pelo menos um veículo por agregado familiar. Contudo esta popularidade introduziu vários problemas, sendo um dos mais visiveis a congestão das vias rodoviárias. Os condutores perdem anualmente vários milhares de euros em combustível desperdiçado e tempo perdido. Outro problema existente, agravado pelo anteriormente descrito, é a poluição gerada pelos veículos. Os poluentes libertados pelos veículos a combustão constituem uma porção considerável dos gases que contribuem negativamente para o efeito de estufa na atmosfera. Aumentar a fluidez do trafego iria ajudar a minimizar os dois problemas descritos. Por último e não menos importante, dezenas de milhares de vidas humanas são perdidas anualmente devido a acidentes rodoviários. Qualquer melhoria que aumente a segurança na realização de manobras rodoviárias tem o potencial de salvar uma quantidade considerável de vidas humanas.

A introdução dos primeiros veículos autónomos está a fornecer a oportunidade de atenuar estes problemas. Actualmente, os sistemas de condução autónoma existentes recolhem informação através de sensores, ie. Proximidade e *Light Detection And Ranging* (LIDAR), montados em pontos estratégicos do veículo, e baseiam as decisões nessa informação com o objectivo de realizar as manobras pretendidas, enquanto mantêm a segurança dos ocupantes intacta. Contudo, nem sempre os sensores têm a precisão necessária em todas as condições de funcionamento de modo a obter informação sobre a qual é possível tomar as melhores decisões. Adicionalmente, os sensores podem inclusivé ter falhas, levando a que estes reportem dados erróneos ou não reportem dados de todo. Deste modo, os sistemas de condução autónoma devem tem em conta estas possibilidades. Uma possível abordagem para garantir a segurança dos ocupantes consiste em o sistema adaptar o seu modo de funcionamente consoante as condições externas e o estado dos seus componentes, tomando medidas preventivas quando não opera em condições ideais, como por exemplo reduzir a velocidade ou aumentar a distância entre os veículos próximos.

Os sistemas de transporte inteligentes são um tópico de pesquisa bastante ativo e um dos principais pontos que estes tratam é a comunicação entre veículos e comunicação en-

tre veículos e infraestrutura. Já existem vários meios de comunicação veicular, sendo o mais promissor baseado no padrão 802.11p e no protocolo *Wireless Access in Vehicular Environments* (WAVE). Esta comunicação entre veículos introduz possibilidades interessantes, como por exemplo a troca de informação entre eles. Isto permitiria colmatar a falha de sensores dos veículos, ou até mesmo a obtenção de informação que o veículo seria incapaz de obter de outro modo, permitindo ao sistema de condução autónoma ter uma melhor percepção do ambiente que permitira a tomada de decisões mais informadas.

Outro aspecto interessante introduzido pela comunicação veicular é a possibilidade de cooperação entre vários veículos. A introdução de cooperação entre veículos autónomos – tornando-os veículos cooperativos – permite o desenvolvimento de aplicações veículares complexas. Essas aplicações podem ser classificadas como aplicações para conforto ou para segurança. Exemplos de aplicações de conforto são informações sobre localização de bombas de gasolina e preço de combustíves , e informação de tráfego. Exemplos de aplicações de segurança incluem pelotões de carros, onde os veículos viajam com distâncias extremamente reduzidas entre eles, sinais de tráfego virtuais, onde os veículos decidem de modo distribuído a ordem pela qual estes atravessam um cruzamento, e aplicações de alerta de serviços de emergência. Contudo, estas aplicações necessitam de garantias de segurança extremamente elevadas devido ao contexto em que operam, isto é, veículos, com passageiros, em estradas públicas e possivelmente com pedestres. É então essencial que a segurança das aplicações seja uma das principais preocupações no desenvolvimento destas, para garantir que o pior cenário seja evitado, isto é, a perda de vidas humanas.

Para a existência de cooperação entre veículos de um modo seguro é necessário que estes se consigam coordenar. Para tal é necessário um protocolo de coordenação entre veículos conhecido pelos mesmos que os auxilie a determinar qual deles é que deve realizar a manobra pretendida.

Nesta dissertação de mestrado é apresentado um protocolo de coordenação de veículos baseado na comunicação entre os mesmos e com suporte de um serviço de *group membership*. O protocolo está permanentemente à espera de novos pedidos de outros veículos e por defeito aceita esses pedidos, a não ser que já exista outro veículo a realizar uma manobra ou que esta nova manobra coloque a hipotese de por em perigo um veículo. Isto permite que para o caso usual, isto é, não existe outro veículo a realizar uma manobra, a manobra não coloque nenhum outro veículo numa situação perigosa, e as mensagens trocadas sejam entregues atempadamente, o pedido seja aceite com recurso a apenas uma ronda de comunicação entre os veículos. O conjunto de veículos que representa a vizinhança de cada um dos veículos do sistema é calculado pelo serviço de *group membership*, informação essa que depois é utilizada no protocolo para determinar se uma determinada manobra pode colocar outros veículos em risco. Foi desenvolvida uma implementação deste protocolo de coordenação em Java, bem como uma aplicação

de teste para verificar o correcto funcionamento do mesmo. As principais decisões tomadas durante a implementação estão descritas em detalhe no decorrer do documento. Foi também desenvolvida uma aplicação de teste que integra tanto o protocolo de coordenação como o serviço de *group membership*. Esta aplicação suporta várias instâncias do protocolo de coordenação, e liga-se a um simulador para demonstrar o funcionamento do protocolo num ambiente cooperativo. Realizou-se também uma avaliação do protocolo. Primeiro efetuou-se uma avaliação do protocolo em três casos de uso para aferir se as propriedades desejadas se verificam, tendo sido obtidos resultados positivos. De seguida, avaliou-se o desempenho do algoritmo segundo duas métricas, em vários cenários com diferentes probabilidades de perda de mensagens: tempo médio para adquirir permissão para realizar uma manobra, e número de tentativas extra necessárias para realizar um total de 250 manobras. Relativamente à primeira das métricas recolhidas, observou-se que o tempo médio para adquirir permissão para realizar uma manobra é na grande maioria dos casos inferior ao tempo que um condutor humano levaria a realizar uma manobra, sendo que apenas nos piores casos testados os tempos se tornam equivalentes, mantendo-se ainda dentro dos valores máximos considerados. Quanto ao número de tentativas extra necessárias para realizar 250 manobras, observou-se que este aumenta exponencialmente consoante a percentagem de falhas de comunicação.

**Palavras-chave:** Veículos Autónomos, Comunicação Veículo-Veículo, Protocolo de coordenação, Serviço de *Membership*

# Abstract

Ever since its introduction, the car has fundamentally changed our society. Its popularity grew tremendously in the early 20th century, and today it is nearly ubiquitous. However, there are several problems related to automobiles, one of the major ones being road congestion. Drivers lose millions of dollars every year in fuel costs and time spent in day to day traffic congestion. Another major problem is emissions from vehicles, forming a significant percentage of greenhouse gas emissions.

Automated driving systems currently rely on their own sensors to gather information from the real-world, and make informed decisions to keep their passengers safe. But sensors may not be sufficiently accurate during all conditions and can even fail, so automated driving systems take this into consideration when controlling the car, leading to larger safety margins.

Vehicle-to-Vehicle communication can enable cooperation between vehicles which, among other things, can be sending or receiving information from other nearby vehicles, increasing the confidence level in the information gathered or even gathering information otherwise not obtainable.

Cooperation opens the door to complex vehicular applications such as road-trains (or platooning) and virtual traffic lights, both of which have the potential to mitigate the problems mentioned before. These applications have tight safety requirements due to the context in which they operate: vehicles, possibly with human occupants, operating on roads traversed by non-autonomous vehicles and pedestrians.

In this MSc Dissertation, we describe the implementation of a vehicular cooperation algorithm backed by both vehicle-to-vehicle communication and a cloud membership service. We then evaluate the implemented algorithm in a cooperative environment to conclude about its correctness, making use of the Robot Operating System middleware to implement a simulation and visualize a maneuver executed using the algorithm.

**Keywords:** Autonomous Vehicles, Vehicle-to-Vehicle Communication, Coordination Protocol, Membership Service

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Ever since the car started being mass-produced, it fundamentally changed the way we move. As countries' economies grew in the 20th century, so did the popularity of the automobile, being a pillar in the lives of most people. According to [17], over 80% of households in developed countries own at least one vehicle. Furthermore, OCDE's Transport Outlook predicts that by 2050 there will be 2 billion vehicles in circulation.

As stated in [5], the transportation sector accounted for 28% of greenhouse gas emissions, of which over half is generated by passenger cars and light-duty trucks. Additionally, this increase in the number of vehicles generates an enormous amount of congestion on the roads. [65] reported that US drivers lost 300 billion dollars in 2016 in fuel costs and time due to congestion, with Los Angeles drivers spending an average of 104 hours in gridlock traffic that year.

Automated driving systems promise to ease these issues and change the transportation outlook. Several companies are currently exploring such autonomous solutions, including Volvo with drive me [21], Google with Waymo [50] (formerly Google car) and Ford through its partnership with Argo AI [49]. The current solutions make use of a variety of sensors and actuators placed in strategic points of the automobile to observe obstacles that might affect the vehicle and perform actions to ensure safety is not jeopardized.

## 1.1 Motivation

Sensory information collected by autonomous driving systems are the only mechanisms these systems have of perceiving the world around them. This information can be incomplete, and therefore we cannot completely rely on it to perform complicated maneuvers.

Vehicle-to-Vehicle (V2V) communication has the potential to allow vehicles to gather information from other nearby vehicles, therefore increasing confidence in the information collected by its sensors, or even gaining access to new information or detecting errors in sensors. This form of communication also introduces the opportunity of V2V coordination and with it coordinated maneuvers.

Several applications can be developed based on V2V communication. Examples of such applications include vehicle road-trains (commonly referred to in literature and henceforth as platoons or platooning) and Virtual Traffic Light (VTL), having both been studied extensively ([7, 23, 28, 29, 13]). Advances in the area of V2V communication allowed us to get near the point of where we can implement such applications, but further work is needed in the area of vehicular cooperation to ensure that such applications are feasible.

Safety must be a consideration however, as communication failures may occur at any point and vehicles must be able to continue to operate safely. They must be able to know if every other nearby vehicle knows the maneuver to perform before starting it, otherwise a rogue vehicle can compromise the safety of all nearby vehicles. With this in mind, vehicular applications need to proactively abort maneuvers when it detects that the necessary safety properties are not guaranteed. Existing vehicular safety applications include traffic-signal-violation warning and emergency electronic brake lights, but future iterations may include applications such as cooperative forward collision warning or stop-sign movement assistance.

Furthermore, even though outside the scope of this MSc Dissertation, pedestrian safety is also extremely important. The first pedestrian death related with autonomous vehicles happened in March of 2018 [39], and is the main example of the consequences of what a catastrophic failure in such vehicles can cause, that is, the loss of human lives.

This poses the question of how can vehicles safely cooperate. As previously described, autonomous vehicles have tight safety requirements that must be met, even when facing failures. Additionally, given the different efforts by several different manufacturers, each in implementing its own Autonomous Driving System, a standard vehicle cooperation protocol.

Nonetheless, in [16], the authors propose a coordination protocol to allow a set of vehicles to coordinate among themselves. The authors also proposes a membership service which is used by the agents running the protocol in the vehicles to know which vehicles are near, and allow that information to be used by the algorithm to check which vehicles will have to be accounted for. We aim to implement the protocol proposed to support vehicle coordination, and perform tests in a vehicular application.

## 1.2 Objectives

The main goal of this MSc Dissertation is to implement a set of protocols to be executed in autonomous vehicles in order to safely perform coordinated maneuvers in an environment comprised of autonomous vehicles. In order to achieve this goal, we will break down the main objective into two separate objectives.

The first objective consists of designing and implementing the agent coordination pro-

tocol, and integrating it with a cloud-based membership service.

The second objective naturally focuses on the evaluation of our work and the complete system integrated with the membership service. A scenario will also be developed in order to enable the visualization of a maneuver being coordinated with the aid of the coordination protocol developed.

## 1.3 Document Structure

The remainder of this document will be structured as follows:

- In **Chapter 2** we will present related work, mainly in the areas of consensus, wireless mediums, V2V Communication and cooperation
- In **Chapter 3** an overview of the coordination protocol will be provided, as well as the properties it aims to ensure and the assumptions made about the system
- **Chapter 4** will describe the development of the solution, our implementation and the decisions made throughout the development
- **Chapter 5** illustrates the integration process between the coordination protocol and the membership service, as well as the simulation scenario developed for visualization of a maneuver coordinated with the aid of the coordination protocol developed
- **Chapter 6** presents the evaluation process and its results

Please note that this dissertation has been accepted as a communication at $11^{th}$ edition of INForum.

# Chapter 2

# Context and Related Work

In this chapter we will list and give a brief overview of a few essential concepts and the state of the art related to our work. The concept of autonomous vehicles is expanded upon, as well as a subset of vehicular applications based on V2V communication. How that communication can be achieved is also expanded upon, as well as the concepts of group membership and virtual synchrony as abstractions to better deal with consensus problems.

## 2.1 Autonomous vehicles and automated driving systems

In Section 1.1 it is shown that autonomous vehicles and automated driving systems have the potential to be extremely useful in reducing road congestion, emissions and fuel usage. Autonomous vehicles are those that are capable of intelligent motion and action without requiring either a guide to follow or teleoperator control [22]. Automated driving systems are a conjunction of several automated systems working together in order to allow a vehicle to autonomously drive itself.

Autonomous vehicles rely on sensors to perceive the world and build their understanding of it. With this understanding, autonomous vehicles are able to make informed decisions adequate to the environment they are operating in.

Some ethical questions can be raised about the topic, such as ”Who is accountable in the event of an accident? The driver or the manufacturer?” and ”Can the information generated by the vehicles be used to identify individuals?”

The SAE International has classified autonomous driving systems in five levels [2], presented in Table 2.1. Most currently available vehicles with autonomous driving features operate on either Level 1 or Level 2. For instance, an Adaptive Cruise Control application lies in Level 1, while Tesla’s Autopilot [42] lies in Level 2. It is clear that automated driving systems are on the range of levels 3 – 5.

| | | |
|---|---|---|
| Level 1 | Driver Assistance | The system helps with some basic driving functions like accelerating, braking and steering, but most of the input is still the responsibility of the driver |
| Level 2 | Partial Automation | The system is capable of handling basic driving function by itself in some conditions, but the driver is still responsible for monitoring the environment |
| Level 3 | Conditional Automation | The system can fully handle basic driving functions and monitors the environment by itself, but the driver maintains tactical awareness in case the system needs a fallback |
| Level 4 | High Automation | The system can fully handle basic driving functions, monitors the environment and maintains tactical awareness in some driving modes |
| Level 5 | Full Automation | The system performs the same tasks as Level 4 Automation, but does so for all driving modes |

Table 2.1: Levels of Automation for Driving Systems according to SAE International [2]

### 2.1.1 Cooperative vehicles

Also described in [2] is the definition of cooperative vehicles, that is, a subset of autonomous vehicles that depend on outside entities to perform their autonomous driving duties. This does not imply that these vehicles are not able to use their own sensors in case they encounter communication errors.

By sharing information and decisions, we can improve traffic flow and safety by coordinating the decisions among vehicles. Information can also be shared with infrastructure, for instance with transport management systems such as traffic signals.

This brings several benefits, such as the potential to reduce the number and severity of crashes, improve the efficiency of the road network and reduce travel time.

Cooperative vehicle also raise security concerns, most notably the authenticity of the information.

## 2.2 Inter-vehicle communication applications

Here we will talk about the main application types described in the literature. Inter-Vehicle Communication applications are in one of two categories [67]:

- **Comfort Applications** - applications that improve passenger comfort and traffic efficiency or optimizes a route to a destination
- **Safety Applications** - applications that increase passenger safety by exchanging safety relevant information

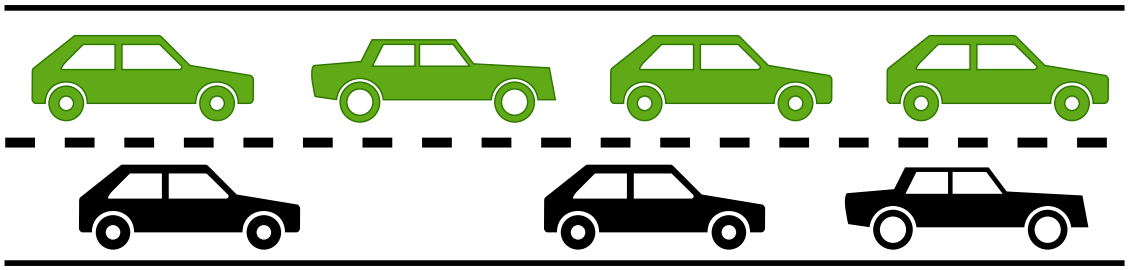


Figure 2.1: Example of a platoon of vehicles (in green) travelling next to regular, non-cooperative vehicles (in black)

Applications that fit in the comfort category include gas station location and price information applications, restaurant location and price information and traffic congestion information. The application we discuss in the following sections can all be considered safety applications.

### 2.2.1 Platooning

Platooning [13] is defined as a collection of vehicles that travel together, actively coordinated in formation. Platooning can improve safety, traffic flow and reduce fuel costs.

Since vehicles travel with short inter-vehicle distances, the lead vehicle in the platoon is effectively "splitting" the air as it travels, creating a low aerodynamic drag area behind it that the following vehicles take advantage of. The reduced inter-vehicle distance also means that the platoon takes less space on the road than if the vehicles were not traveling together. This also means that it is not subject to sudden accelerations, further helping with saving fuel. In [7] the authors study fuel economy in platoons composed of trucks, and conclude that a fuel saving of 4.7% to 7.7% can be achieved, depending on inter-vehicle distances. Since the vehicles are coordinated, when one of the vehicles needs to perform an emergency braking maneuver, the following vehicles can immediately react, not being subject to human reaction times, hence reducing braking distances in emergency situations. This specific point is further discussed in Section 2.2.6.

Figure 2.1 shows an example of how a platoon would travel, compared to non-cooperative vehicles. Please notice the reduced inter-vehicle distances.

### 2.2.2 Lane merging assistance

Lane merging has several specific instances (i.e. joining a platoon). In a lane merge scenario, the merging vehicle has to assure that its maneuver will not be a safety hazard to other vehicles. This can be achieved by the merging vehicle communicating with other nearby vehicles to ensure all vehicles know its intentions and enable the merge.

When a new vehicle wants to join a platoon, it coordinates with the members of the platoon the location in which it will join. After this negotiation, the vehicles in the platoon

will generate a gap at the agreed location, and the new vehicle will merge into the platoon and resume its normal operation.

The reverse happens when a vehicle wants to leave a platoon. It communicates its intention to both the platoon and nearby vehicles, and when the vehicle safely merges into a different lane, the existing platoon closes the gap left by the leaving vehicle.

### 2.2.3 Intersection Crossing Assistance

According to the US Department of Transportation Federal Highway Administration, more than 50% of the combined total of fatal crashes and crashes with injuries occur at or near intersections [55]. Automating Intersection crossing can therefore undoubtedly provide safety benefits. Knowing nearby vehicles, we can coordinate who crosses the intersection, and when they do it.

### 2.2.4 Virtual Traffic Lights

While not a maneuver in itself, VTL's can be considered as an extension to the Intersection Crossing scenario. In a VTL system, the right-of-way is decided in a distributed manner, with the use of V2V communication. This self-organizing traffic scheme does not need infrastructure at intersections.

According to [29], only 20% of intersections in cities are signalized. Additionally, the authors argue that introducing VTL's will reduce the average commute by 30%. This would be a significant help in reducing congestion in large cities, which usually suffer from crippling traffic and congestion issues.

In [28] the authors show that introducing VTL's in areas with high traffic density could reduce carbon emissions by as much as 20%.

VTL's, with support from vehicular cooperation, are an efficient solution for todays traffic overflowing cities, and related problems created by congestion.

### 2.2.5 Roundabout assistance

Over the last few couple of decades, roundabouts have become a popular alternative to intersections. They reduce or nearly eliminate head-on and driver-side crashes, and have reduced energy consumption and maintenance costs. However, drivers do not always make good use of roundabouts, and when such happens the adjacent areas may become subject to traffic jams [3].

Literature exists on roundabout applications, specifically management, speed and flow control ([11, 69, 58]), but not much work exists of completely autonomous vehicles navigating and cooperating in roundabouts. In [56] the authors study how trajectories can be generated from roundabout parameters such as diameter and center point coordinates and explore control techniques that can implement lateral control of autonomous vehicles. In

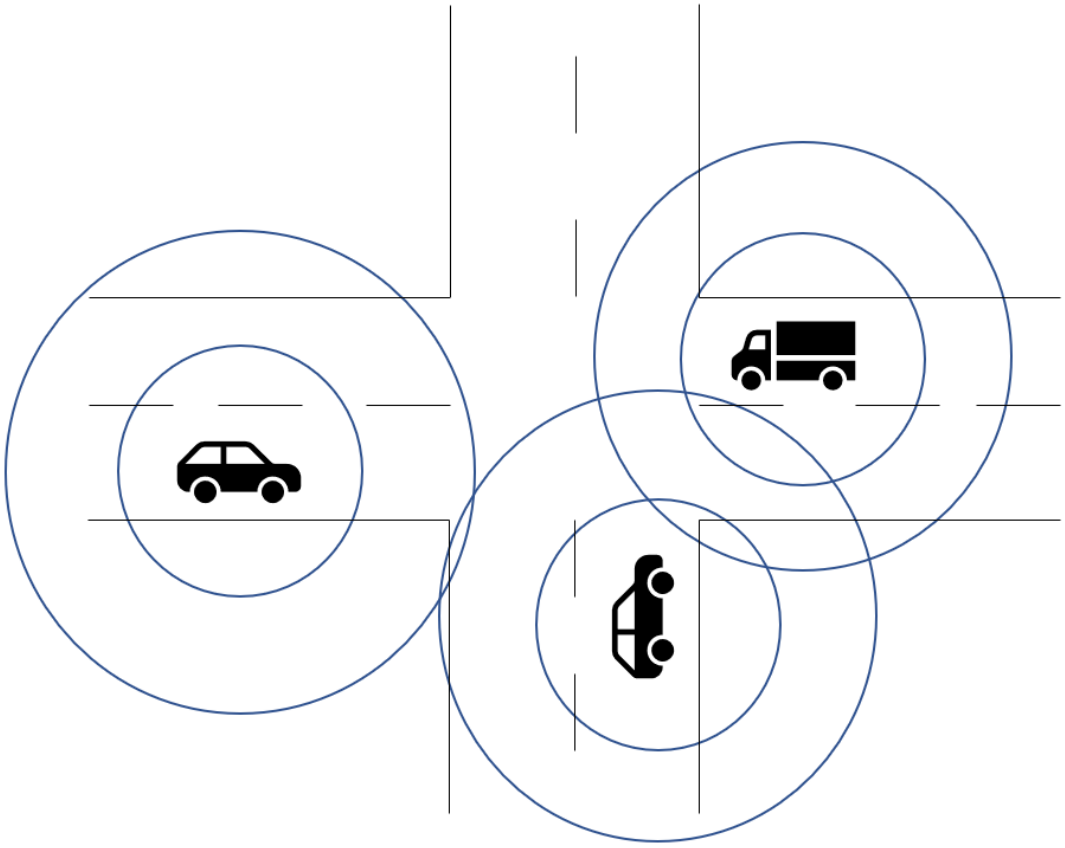


Figure 2.2: Example of an intersection with vehicles working in a Virtual Traffic Light application

[59] the authors propose a fuzzy logic controller for lateral control in roundabouts, which generates trajectories based on roundabout parameters (diameter, center coordinates).

### 2.2.6 Cooperative Collision Avoidance

Millions of traffic accidents occur every year. In 2016, there were 7.2 million traffic accidents in the United States of America, with 3.1 million people injured and 37461 killed ([33]). The leading cause was the driver's behavior.

The inability of drivers to react in time can lead to chain collisions. Drivers rely on the tail lights of other cars to determine emergency situations, which may not be enough. This visual feedback loop is not enough to guarantee collision avoidance.

Cooperative collision avoidance makes use of V2V communication to quickly and safely react to emergency situations. Upon detecting an emergency, the vehicle can send a message informing nearby vehicles of such emergency, and these vehicles can immediately take preventive actions, not being subject to the drivers reaction time.

### 2.2.7 Emergency Services

Assuming vehicles are connected and exchanging information, and that vehicles are equipped to detect when a crash occurs and retains connectivity after the crash, it is trivial to disseminate this information across the network.

A simple example of an interesting application is one that detects when a crash occurs and immediately calls emergency services, giving crucial information such as the number of vehicles involved and their positions, to emergency services. They will then be able to assist passengers with a shorter response time, increasing the chances of survival if the passengers suffered life threatening injuries.

In [52] the authors provide a comprehensive outlook on how emergency services could benefit from vehicular communication, either directly with each other in the form of V2V communication or using Vehicle-to-Infrastructure (V2I) communication.

## 2.3 MANETs and VANETs

A Mobile Ad-hoc NETwork (MANET) [19] is a network comprised of mobile nodes, forming a mesh network with a dynamic topology. Given that there is no fixed infrastructure in the network, nodes are expected to assist by adopting packet routing duties. Their dynamic nature also allows them to self-heal. Nodes connect to their neighbors, and this set changes as nodes move through the area, changing the topology. As long as the network is not partitioned, all nodes can be reached.

Given how well MANETs adapt to dynamic environments, they are perfect candidates to use in vehicular communication. A VANET(see Figure2.3) [70], is a specific instance of a MANETs in which nodes are vehicles. VANETs can enable V2V or V2I communication using either Short Range Radio Technology standards such as IEEE 802.11p [43] or IEEE 802.15.4 [1], or celular technologies such as LTE. Most applications use the Wireless Access in Vehicular Environments (WAVE) standard, based on IEEE 802.11p. This protocol uses Dedicated Short Range Communications (DSRC) [25, 68], which has a range of 1 km with transmission rates varying from 3 Mbps to 27 Mbps range and can withstand vehicle velocities up to 260 Km/h. Messages are exchanged over the medium using the WAVE Short Message Protocol [66] described in standard IEEE1609. This protocol supports high priority, time sensitive communications [47, 36], but does not guarantee message delivery. In [38] the authors show that the delay of control messages with the highest priority remains in the order of tens of milliseconds, and that delay only becomes excessive when traffic reached 1000 packets per second.

Traditional MANET routing protocols such as Ad Hoc Distance Vector [57] require an explicit route establishment phase prior to data transmission taking place. This means that applications that require low delivery latencies may not use such protocols. Another factor that makes such protocols not suited for vehicular safety applications is that receivers

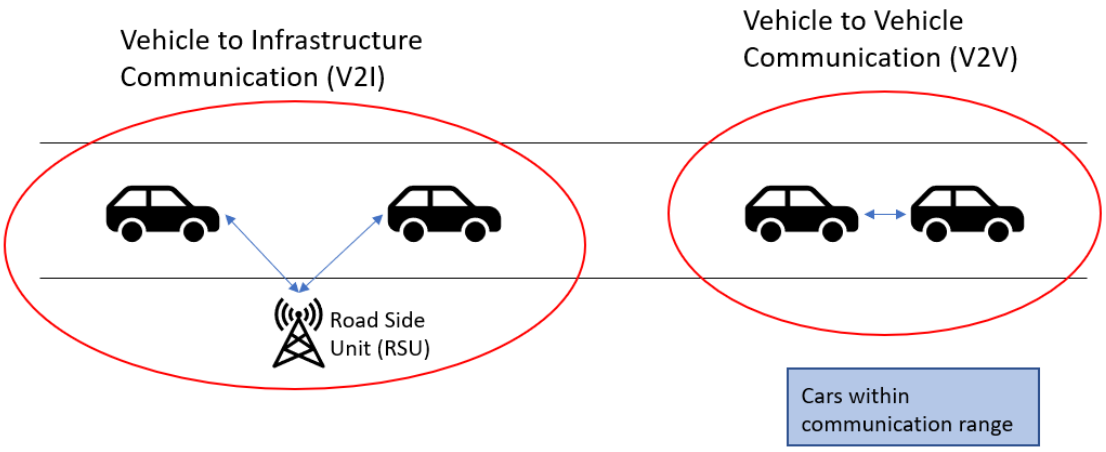



Figure 2.3: Example of vehicles communicating within a VANET

may be unknown. Considering the two points above, broadcast oriented protocols are preferred. Context-aware broadcast can further increase efficiency, by applying direction-aware broadcast forwarding. This implies that a packet is forwarded based on the direction it was received (i.e. if a node receives a packet coming from North it is forwarded to the South). This approach has the benefit of reducing the number of messages sent, reducing the chances of collisions. A high collision rate causes reduced delivery-rates and increases latency.

In [44] the authors present the operational concept of 5.9Ghz DSRC-based vehicular-safety communication, and propose a set of protocols to address the issues found. In [15], the authors implemented a cooperative collision avoidance application based on an intelligent broadcast primitive with direction aware broadcast forwarding and show that for packet error rates up to 50% crash performance is not affected. In [68] the authors also implement a cooperative collision warning application based on a location-based broadcast protocol. Some Location-Aided Routing protocols are shown in [46]. In [4] a more broad overview is presented, categorizing protocols into ”proactive” or ”reactive”.

## 2.4 Virtual synchrony and group membership

Distributed Consensus has been heavily discussed in literature ([35, 53, 30, 61, 12, 60]). It considers the selection of a single value from a set of values proposed by members of a group. The selected solution is required to be reached within a bounded time. There are a number of impossibility results related to distributed consensus ([31, 27, 32]). It is shown by the authors in [51] that the presence of communication failures makes it impossible to deterministically reach consensus.

As mentioned in Section 2.3, VANETs uses V2V communication, and the former are designed to adapt to dynamic environments. However, in a vehicular environment,

communication is inherently prone to failures. When these communication failures are too frequent, vehicles can fail to reach consensus for an unbounded time, which poses a safety threat.

The existing literature on consensus algorithms with real-time requirements generally assume timed and reliable communication [40, 6]. In this work, is is assumed that messages arrive within a known bounded time, but that there is no bound on the number of consecutive message omissions. It is therefore possible to detect message omissions.

Automated driving systems with V2V communication are safety-critical applications, therefore being paramount that faults do not lead to catastrophic failures. Alpern and Schneider define liveness in [8] as "A liveness property stipulates that a good thing happens during execution". In distributed consensus, the good thing the execution tries to achieve is the selection of a single value. However, in the scenario of V2V communication, where communication failures cannot be ruled out, this implies that the system will wait for a consensus. Since we assume that there is no bound to the number of consecutively dropped messages, this effectively means that nodes can try to reach consensus for an indeterminate amount of time. Thus we need to ensure that, if no consensus is reached within a certain upper bound, the execution must continue with no consensus, with vehicles possibly aborting intended maneuvers and relying only on their sensors for example.

In this context, we can consider topology changes to be transient failures since they lead to inconsistencies. One way to achieve this is through virtual synchrony. Birman et al. describe virtual synchrony in [14]: "It will appear to any observer that all processes observed the same events in the same order. This applies not just to message delivery events, but also to failures, recoveries, and group membership changes". From this, we can see that two key concepts are needed to create virtually synchronous executions (ie. the notion of a group and an atomic multicast primitive).

Group membership services act as an abstraction layer for consensus problems. As mentioned before, most work was based around the node crash fault model, but there has been significant work on tackling network partitions. Examples of such work include Transis [9], Totem [54], Moshe [45] and Jgroups [10]. In [24] the authors also present a Group Service for dynamic networks. In the context of vehicular cooperation, such services maintain a view of which cars are neighbors and ensure that cars have consistent views on their neighbors. Lim and Conan address the problem of group membership in MANETs in [48]. Other approaches to form group membership exist. In [63] Cahill and Slot take a hardware-based approach, in which vehicles make use of laser-scanners to retrieve information regarding empty areas around themselves, and share this information with other vehicles, attaining precise membership information. In [26] the authors make the distinction between safe and unsafe disagreement, the latter being when nodes decide on different non-empty views, and propose a decision algorithm based on synchronous

rounds, unreliable communication and unreliable oracles. A more broad overview of group communication services is presented in [20].

# Chapter 3

# Protocol Analysis

In this chapter an overview of the vehicle coordination protocol that was proposed to be implemented and which is described in [16] will be provided, along with the models used, assumptions, limitations, and properties that should be assured.

The agent coordination protocol aims to allow agents (ie. vehicles) to obtain support or permission from other agents for performing a maneuver, which will allow the maneuver to be done in a more efficient way than if done without any coordination.

## 3.1 Assumptions

The following assumptions are made:

- **Membership**
  - Accurate membership information may be available
  - Membership information has a defined validity period, which allows detecting when it is no longer valid
- **On-board Sensors**
  - On-board sensor information is insufficient to perform autonomous maneuvers efficiently
  - Facing communication failures, on-board sensor information is sufficient for eventually performing a maneuver in a safe manner

## 3.2 Models

### 3.2.1 Communication Model

- Agents may communicate with other agents within Z1[1] of each other

[1] Maximum Communication Range (Agent to Agent)

- The network is fully connected between agents and the infrastructure
- V2V Communication and V2I Communication are asynchronous
- The number of consecutive message omissions is unbounded

This communication model implies that any agent can communicate with nearby agents and with any infrastructure process, and that communication delays are unbounded. It also implies that the protocol must be able to deal with these unbounded delays and message omissions, while achieving the desired goals (described in Section 3.3) Based on the aforementioned assumptions, one can conclude that it is not possible to guarantee that all needed remote information will be received and that there will be an indication that all such information was received.

### 3.2.2 System Model

- The number of agent processes is bounded, but its bound is unknown
- Agent processes are also physical entities, with physical properties
- The number of infrastructure processes is bounded but unknown
- All processes have access to a local clock perfectly synchronized with a global time

The key point in the system model for the protocol is the existence of local clocks synchronized with a global time, which allows us to rely on timestamps. This can be achieved in vehicles through GPS, for example. Based on globally synchronized timestamps it becomes possible, for instance, to determine if a certain membership information, created at some time t, is still valid. It also allows to determine the delay of transmitted messages and discard the ones that took too long to arrive and therefore may contain outdated information.

### 3.2.3 Fault Model

- Agent processes do not fail arbitrarily
- Agent processes may crash and their physical instantiation will leave the system in a finite amount of time
- Infrastructure processes do not fail arbitrarily
- Infrastructure processes may crash

As specified by the fault model, byzantine failures are not handled by the protocol.

## 3.3 Properties

The protocol aims at guaranteeing two basic properties:

- **Safety**
  - Safe maneuver - Regardless of the presence of failures, no two vehicles perform maneuvers concurrently
- **Liveness**
  - Timely maneuver - In the absence of failures, a vehicle intending to do a maneuver will do said maneuver in a bounded amount of time

Additionally, from a practical point of view, the protocol is not particularly useful if the agent needs a larger time frame to execute a maneuver than when in a situation where no cooperation is possible. From this observation, it is possible to extrapolate an additional desirable property:

- **Performance** - The average time that it takes for an agent intending to do a maneuver to execute said maneuver is, at most, the same as when the agent is not able to interact with other agents and executes the maneuver using only information provided by its local sensors

## 3.4 Protocol Architecture

In this Section a detailed description of the protocol will be provided. A pseudocode description of the protocol is also provided in Annex A

### 3.4.1 Protocol Parameters

The protocol has some configurable parameters that are used throughout the protocol, described in Table 3.1.

| | |
|---|---|
| $T_D$ | Upper bound on transmission delay |
| $T_A$ | Period of agent registry update |
| $T_M$ | Period of membership protocol |
| $T_{MAN}$ | Upper bound on maneuver execution time |

Table 3.1: Parameters used in the agent coordination protocol

$T_D$ specifies the upper bound on transmission delay the coordination protocol tolerates. Messages which are deemed to have had a transmission delay larger than this value are dropped. $T_A$ is the value used for the timer that triggers the job to update the agent's

information in the membership service. $T_M$ is used to calculate if a membership received by the membership service is still valid. Finally, $T_{MAN}$ is the maximum amount of time a vehicle has to perform a maneuver, and is used to calculate the maximum amount of time a grant can be active.

### 3.4.2 Agent States

The agent (i.e., the protocol executing in the scope of an agent), can be in one of six different states, depending on the progress in the protocol execution since a request for permission (grant) for doing a maneuver is sent to nearby vehicles (indicated in the currently valid membership) until this permission is obtained and the maneuver is done.

- **NORMAL** - When no maneuver is currently being executed
- **GRANT** - When the current agent has given a grant to a nearby agent to perform its maneuver
- **TRYGET** - When the current agent is trying to obtain a grant from nearby agents to execute its maneuver
- **GRANTGET** - When the current agent has given a grant, and intends to request one after the current grant has expired
- **GET** - When the current agent is awaiting the response for the request made
- **EXECUTE** - When the current agent is executing the desired maneuver

Agents state transitions are indicated in Figure 3.1.

### 3.4.3 Message Types

The messages can have only one of the following four types:

- **GRANT** - Sent when the agent accepts a maneuver request from another agent
- **DENY** - Similar to **GRANT**, but sent when the request is denied
- **GET** - Sent when the sender requests a grant
- **RELEASE** - Sent when the sender explicitly releases a grant

When these messages are received, they are processed as specified in Section 3.4.8.

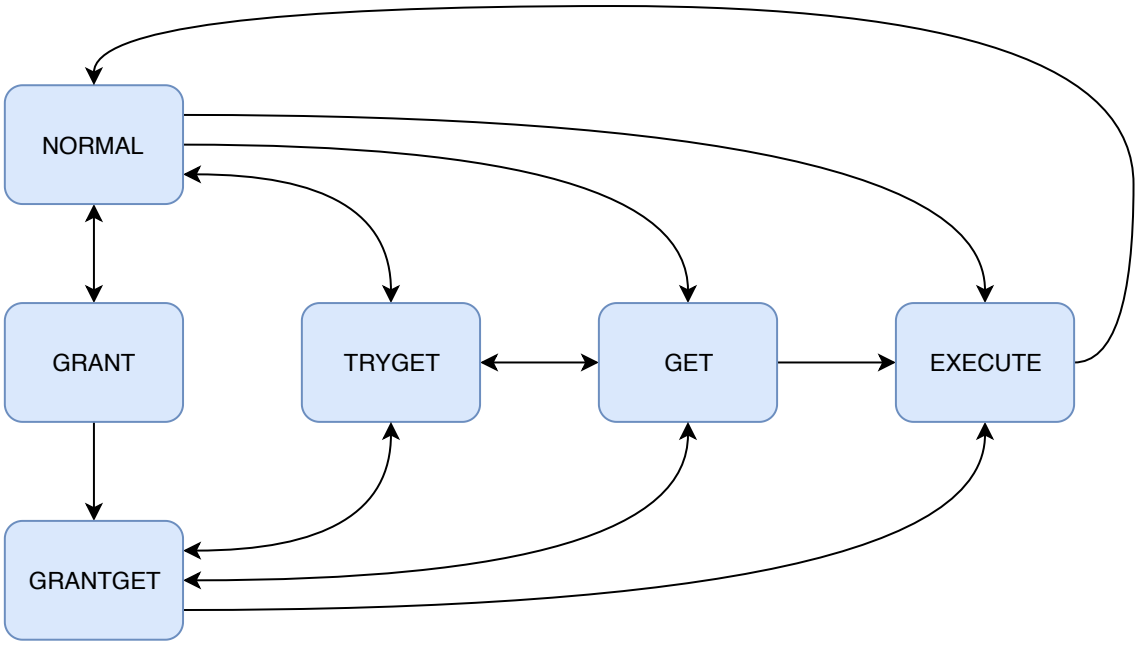



Figure 3.1: Existing Agent States and possible transitions between them

### 3.4.4 Timers

The protocol has three internal timers it uses to schedule certain procedures:

- **T_UPDATE** - executes a procedure that updates the agent information stored in the membership service. Runs with a period of $T_A$ while the agent is active
- **T_GRANT** - this timer is started when the agent gives another agent a grant, and when fired executes a procedure where it will revert its state to **NORMAL**, and if a grant request is pending will execute *tryManeuver* as described in Section 3.4.7
- **T_RETRY** - started when a grant request is sent, when triggered this timer will execute a procedure to retry the grant request. This timer is cancelled when the agent receives a response from all expected agents in a timely manner

### 3.4.5 Maneuver Identification

Each maneuver is identified by a **Maneuver Tag**. These tags consist of the ID of the agent that requested the maneuver, and the timestamp of when the request was created. The timestamp of the maneuver is used to decide who has priority in case a conflict exists (ie. two grant requests received within a short amount of time).

### 3.4.6 Initialization

Upon initialization, the agent will be in state **NORMAL**. From this point forward, the agent will listen and process arriving messages from other agents. It will also be able to

try maneuvers, requesting grants from nearby relevant agents.

### 3.4.7 Trying to execute a maneuver

When an agent tries to execute a maneuver, a procedure *tryManeuver* is executed. In this procedure, the agent will check wether it is currently in a state where it is possible to start a maneuver, these states being **NORMAL** or **GRANT**. If in one of these states, the agent will create a new **Maneuver Tag**.

Following this, the agent will do another check to confirm if it is in a state where it can request a grant, these being **NORMAL** or **TRYGET**. If in any of these two states, the agent will update its membership information, verify its validity and, if there exists a maneuver opportunity (**MO**), will proceed to send messages to agents in its membership requesting a grant to perform the maneuver. The maneuver opportunity is an information sent by the membership service to indicate if an agent can initiate a maneuver. Since the membership service has a global vision of the system, it is better informed to inform if the conditions allow for a maneuver to be executed.

After sending the grant request, the agent will start a timer **T_RETRY**. When this timer fires, the agent will resend the grant request, change status to **TRYGET** if its status is **GET**, and will execute the procedure *tryManeuver* again, regardless of its previous state. If its membership is empty, meaning there are no nearby agents that might impact the maneuver, the agent will immediately execute the maneuver.

If instead the agent is not in a state where a grant request is possible, but is in state **GRANT**, meaning another agent has an active grant, the agent will change his state to **GRANTGET** and will retry to execute the *tryManeuver* procedure when the current grant timer terminates.

### 3.4.8 Message Processing

The agent is perpetually listening for new messages. When a new message arrives, it must be processed. The first step in this process is to check wether the message is timely, accomplished by comparing the message timestamp and the reception timestamp. If the difference between the timestamps is less than $T_D$, the message is deemed timely and processing continues, otherwise it is deemed ”not timely” and processing of that message halts with the message being discarded. This complies with the assumption that messages arrive within a bounded amount of time.

After ensuring that the message is timely, the type of the message is checked to determine how further processing should be carried out. If the message type is either **GRANT** or **DENY** and agent is in state **GET**, meaning the message received is a reply to a grant request previously sent by the agent, processing will continue by removing the agent that sent the message from the set of expected responses, and check wether this is the last

expected message. If that is the case, **T_RETRY** timer will be stopped and the agent will check if any of the received responses is of type **DENY**. If at least one **DENY** message was received, the agent will change state to **TRYGET**, send a **RELEASE** message and restart timer **T_RETRY**. If all messages are of type **GRANT**, the agent will change status to **EXECUTE** and execute the maneuver requested, after which it will return to state **NORMAL**.

If instead, the message type is **GET**, the agent will check if the following are true:

- there is no conflict between this agent and the agent that sent the message for the period $2T_D + T_{MAN}$
- the agent is in one of the following states:
  - **NORMAL**
  - **TRYGET**
  - **GET** and the agent's maneuver tag does not precede the tag received in the message
  - **GRANT** or **GRANTGET** and the agent has given a grant to the agent that sent the message

If there is no conflict between the agents and the receiving agent is in one of the listed states, message processing continues, otherwise a **DENY** message is sent to the sender. When processing continues, if the agent is in state **NORMAL**, it will change state to **GRANT**, else if it is in states **GET** or **TRYGET** it will stop its timer **T_RETRY**, change status to **GRANTGET** and broadcast a **RELEASE** message. In both cases, a **GRANT** message is sent, notifying the acceptance of the request, and the timer **T_GRANT** started.

The final possible case is if the message type is **RELEASE**, the agent is in state **GRANT** or **GRANTGET**, and has given a grant to the sender. In this case, the agent will stop the timer **T_GRANT**, will revoke the grant given out and if currently in state **GRANTGET**, will change state to **TRYGET** and call procedure *tryManeuver*, otherwise it will change state to **NORMAL**.

## 3.5 Considerations Regarding Communication Failures

Whenever a maneuver request is initiated timer **T_RETRY** is started, as mentioned previously. Also mentioned previously is that when the request does not receive all necessary responses from nearby agents, or when all answers are received but one of them is a **DENY** message, a new communication round will be needed to retry the maneuver request. It is also clear that to successfully coordinate a maneuver at least $2 * (n - 1)$ messages need to be exchanged, where $n$ is the number of vehicles to coordinate.

If a single message in a maneuver request is dropped, the protocol will not progress until timer **T_RETRY** is fired, at which point a **RELEASE** message will be sent to all agents, and the request will be started again. Since the value of timer **T_RETRY** is $2T_D$, a retry implies an implicit increase in the time needed to achieve a grant to perform a maneuver of at least $2T_D$.

Additionally, since the number of messages needed to be exchanged increases exponentially, so will the amount of retries needed when facing communication failures.

# Chapter 4

# Implementation

In this chapter, we will walk through our decisions when implementing the agent coordination protocol, the available options and how we proceeded along with our reasoning.

## 4.1 Compatibility

One of the main objectives of this project is to provide a working proof of concept implementation of the protocol working in tandem with the Membership Service. As such, there must be a degree of compatibility between the implementations of the coordination protocol and the membership service. The communication between the coordination protocol and the membership service is done through a library specific to the membership service implementation. This obviously implies that the membership client library must be usable by the protocol implementation. The safest way of assuring this is by both being implemented in the same programming language, and the library providing a stable API for use in the protocol.

## 4.2 Key Decisions

Before actually starting to implement the protocol, a few key decision needed to be made:

- Middleware
- Communication among agents
- Programming language
- Tools

Regarding Middleware, our analysis focused on the robotics middleware Robot Operating System [34]. ROS is, contrary to what its name implies, not an Operating System

but rather a set of tools and frameworks for robot software development. It provides hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Message-passing functionality of ROS is implemented as a Publisher/Subscriber model, where executables can listen and publish data to specific topics. The latest ROS distribution at the time of writing, Melodic Morenia, provides bindings for Python 2.7 and C++. A port of the ROS distribution Kinetic Kame in Java called rosjava is also available, although with unofficial support.

Communication among agents is, as described earlier in 3.1, bounded in delay, but not bounded in number of omissions. Initially, it was hypothesized that ROS' message-passing facilities could be used to establish communication among agents. However, due to the Publish/Subscriber nature of the model, this was not a good fit with the assumptions made in the protocol. Therefore, it was decided that using simple UDP Datagram based communication was the best option, due to it perfectly fitting our assumptions (Connectionless and with no handshaking, thus not providing guarantees of delivery, ordering or duplicate protection).

With regards to programming language and tooling, which obviously will depend on the language selected, two approaches were investigated:

- A C++ based approach leveraging the native ROS bindings
- A Java based approach leveraging rosjava

Each of these approaches, as well as their motivation, will be explained in the following sections.

## 4.3 C++ Approach

The first approach to development was to have simulated agents execute inside the ROS runtime, each running an instance of the protocol. Communication with a visualization platform would be handled by the Publish/Subscriber system part of the ROS runtime. As for communication among agents, it would be done using standard UDP Sockets of the host operating system where the ROS runtime is running.

Additionally, a spawner was planned to ease testing, allowing to dynamically spawn agents and configure them to start maneuver requests based on certain predefined conditions. This allows easier testing of the protocol by effortlessly preparing the environment, taking into account the desired configuration.

As mentioned in Section 4.1, the protocol will have to interact with the Membership Service through a library. To ease this integration, the library should use the same language as the protocol. The Membership Service makes use of Apache Zookeeper, which provides Java and C bindings.

Given the above points, a decision was made to implement the agent coordination protocol in C++, in order to maximize compatibility between all parts and due to the sensitive nature of the application, namely requiring efficient processing of the received messages.

Significant difficulties were found when working on this approach. The most disruptive being the unfamiliarity with C++. This meant significant effort was needed to achieve most of the desired functionalities. Additionally, implementation of the membership service was facing similar issues due to unfamiliarity with the language, as well as a general lack of documentation for the C binding for Apache Zookeeper, leading to several workarounds to get a working library for use in the protocol.

These factors lead to a discussion regarding changing from implementing the protocol in C++ to a different language. Since Apache Zookeeper also provides Java bindings, a decision was made to drop this approach and instead move over to a Java-based approach, using rosjava instead of native ROS.

Although this implementation did not fully materialize, the time spent analyzing the protocol and possible solution for its implementation proved immensely valuable when working on the Java-based implementation.

## 4.4 Java-based Approach

Before reimplementing the protocol in Java, a new structure for the implementation of the coordination protocol was drafted.

To ease implementation and maintainability, it was decided to separate several parts into their own codebases, and to use a build system to allow different parts to work together. Maven was the selected build system, due to it being the standard build tool used with Java development.

The protocol implementation was planned in a manner that would easily allow it to be embedded in other applications. This decision allows for the retention of the concept for the Spawner to ease testing, as well as allow the creation of other testing clients if we so desire. In this new approach ROS is still used, however its scope is much smaller, being used only for its message passing abilities. Agents are emulated outside of the ROS runtime, in a Java-based application, similar to the Spawner mentioned. The visualization components, and the integration of the protocol with both components and with the Membership Service is described in detail in Section 5.

The project was split into the following codebases:

- **Common** - A set of utilities and structures used mostly throughout the project. Includes domain objects that represent core structures (agent, membership, message, etc)
- **Protocol** - the main protocol codebase, where the majority of the protocol logic is

- **Protocol-Spawner** - The Spawner used for testing, as well as a simple Wrapper around the protocol that can be used standalone
- **Membership-Client** - The library used for communication with the Membership Service

Please note that the Membership-Client library is not developed within this project, but it does provide a stable interface initially described in the reference material for this work.

### 4.4.1 Common Module

The common module, as described previously, contains utilities and structures used throughout the project. Its goals is to reduce code duplication by aggregating reusable code in a common dependency.

It contains the following domain objects:

- **Coordinates** - Indicates a position in the physical world, in reference to a pre-established origin
- **AgentState** - Used to represent the state of an agent within the physical world (Position, acceleration and velocity)
- **Agent** - Represents a physical agent. Contains the agents' state and how he can be reached
- **ManeuverTag** - Used to identify a maneuver. Contains the timestamp of when it was requested and the identity of the agent that requested it
- **Membership** - Represent a Membership group. Contains the timestamp indicating its validity, and a list of agents that belong to it.

These Domain objects are used throughout both the protocol and the membership service. Implementing these in a separate module and then importing the module as a dependency in other projects helps guaranteeing that all projects use the same objects, and reduces code duplication improving both readability of the code and its ease of maintenance.

Additionally, it also contains some interfaces used by the protocol:

- **AgentProperties** - Defines methods that return physical properties of the Agent, such as position, velocity and acceleration
- **ManeuverCallback** - Defines callback methods invoked when a maneuver reaches a certain point.

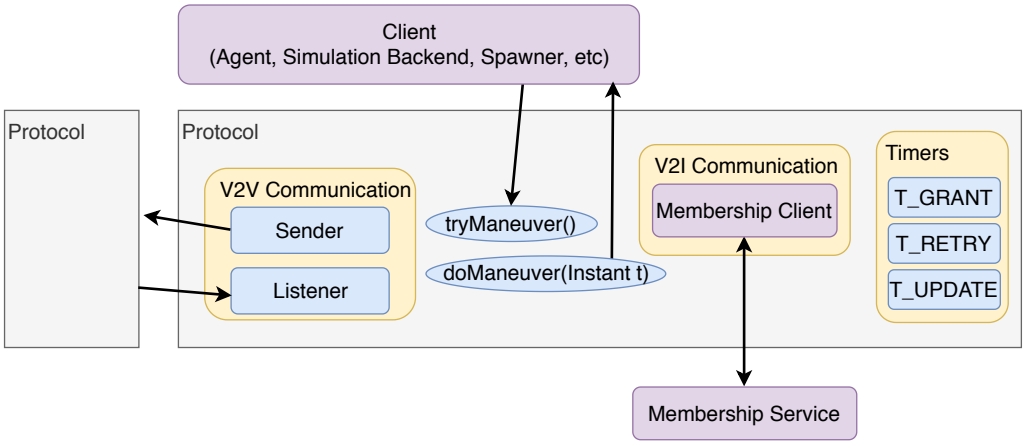



Figure 4.1: Main components of the vehicle coordination protocol implementation

- **MembershipClientInterface** - Implemented by the Membership Service client library, specifies the API that the protocol should use for communication with the service

In summary, all of the interfaces described above are very important for the protocol. AgentProperties is used whenever there is a need to gather information about the physical properties of the agent. The decision to implement AgentProperties as an interface stemmed from the fact that implementation obviously differ depending on the environment we are operating in, and it is not within the scope of the protocol to know all the possible different implementations. ManeuverCallback is used at critical points in the lifecycle of a maneuver to communicate critical information to the agent, usually related to the state of the maneuver. However, it is not the responsibility of the protocol to know what actions should be performed in response to this information. Finally, MembershipClientInterface specifies the methods that the protocol has available to interact with the Membership service, and is implemented by the membership client library.

There is also a **MembershipConnectionException** in this module, used in the MembershipClientInterface. It is thrown in the methods that communicate with the membership service whenever a communication failure arises.

### 4.4.2 Protocol Module

This module is where the majority of the logic regarding the protocol is implemented. Figure 4.1 presents the main components that compose the protocol, as well as the connections to other outside entities.

The V2V communication block contains the two objects responsible for establishing communication with other agents, **Sender** and **Listener**. The Sender class wraps a DatagramSocket object for UDP communication and provides two functions:

- **send(Message, Address)** - Sends the Message to the provided Address

- **multicast(Message, Set<Agents>)** - Broadcasts the Message to each Agent in the provided Set using the send function

The Listener class also wraps a DatagramSocket object, but functions in a very distinct manner. Upon initialization, a new Runnable object is created. After the initialization of the protocol is finished, the Runnable is started in a new thread. The Runnable consists of a while loop in which it will permanently listen for new incoming messages until explicitly requested to stop doing so. Whenever a new message arrives, it is then unserialized into a Message object, and the message processing routine is invoked.

To help testing during development the Listener provides the ability to set a reception success probability, in order to simulate scenarios where communication failures exist. That process is done by discarding messages in a probabilistic manner, according to the probability set.

V2I Communication is, as mentioned before, handled by the MembershipClient library. This library implements the MembershipClientInterface available in the common module. Section 5.1 describes the library in further detail.

All communication with the Membership service is abstracted by this library. Additionally, should a different implementation of the membership service arise, switching implementations should be straight forward for the protocol, as long as its API remains stable.

The timers described in Section 3.4.4 are here implemented. Each of the timers' respective jobs is implemented as a Runnable. They are scheduled through the means of an ExecutorService, more specifically a ScheduledExecutorService. This ScheduledExecutorService holds control over a thread pool comprised of three threads that are used to compute the jobs as needed. Whenever a job is scheduled it returns a Future object. This Future object represents the result of an asynchronous computation, and also exposes procedures to cancel the latter. As such, we save the respective Future object related to the job in memory, in order to be able to cancel the latter, effectively cancelling the corresponding timer.

To interact with the protocol, the host Agent has the public method *tryManeuver()* available. When the host Agent invokes this method, the a new maneuver request is started, performing the actions described in Section 3.4.7. When the possibility to execute a maneuver exists, the protocol will call an internal procedure *doManeuver(Instant t)*. This procedure will invoke the current instance of the ManeuverCallback interface, informing the host Agent of the clearance to perform the requested maneuver until instant *t*. After this instant t, other agents will revoke the clearance given.

### 4.4.3 Protocol Spawner

The idea of a spawner to help with testing has transitioned from the C++ to the Java implementation, and two different spawners are provided in this module.

- **Spawner** - Base spawner used for evaluation
- **Configurable Spawner** - Spawner used for testing interaction with Membership Service

The base spawner was mostly used throughout development to verify correct implementation of the protocol. Its functioning is rather simple:

- Instantiate a predetermined number of protocol instances
- Periodically request a random instance to perform a maneuver for an undetermined amount of time

This spawner was later extended to allow evaluating the protocol by performing a couple of extra steps:

- Upon completing a pre-defined number of maneuvers, no further maneuvers are requested
- Shows statistics gathered during execution, including total number of requests, total number of maneuvers completed, and time taken to complete each maneuver

In order to allow this spawner to work without the need for other components, several dummy implementations of interfaces are provided:

- **DummyMembershipClient** - Simply returns all agents instantiated by the spawner
- **StaticAgentProperties** - Returns a set of static agent properties when queried
- **MeasurementsManeuverCallback** - Used when gathering metrics from the protocol instances for evaluation

# Chapter 5

# Integration

In this chapter a description of how the coordination protocol was integrated with the membership service, as well as the application developed to test both the coordination protocol and the membership service with a scenario in a simulator.

## 5.1 Integration with Membership Service

As described in Section 4.4.2, interaction between the coordination protocol and the membership service is done through the membership-client library.

This library provides two methods for use in the protocol:

- **storeAgentRegistry(Agent)** - this method is used to store information regarding the current agent in the membership service

- **getMembershipRegistry()** - used to request the current membership information relevant to the current agent

The method **storeAgentRegistry** receives a parameter of type Agent, described previously in Section 4.4.1, and stores the state of the agent in the membership service. This method is periodically called as part of the job run whenever timer T_UPDATE is fired. On the other hand the result of method **getMembershipRegistry** is a Membership object containing the relevant agents for the current maneuver and the membership validity. This method is invocated whenever an agent intends to perform a maneuver, as part of the *tryManeuver* procedure.

The use of this library abstracts the whole communication structure with the membership service, making it considerably easier to use the service. The actual library implementation details are unknown to the protocol.

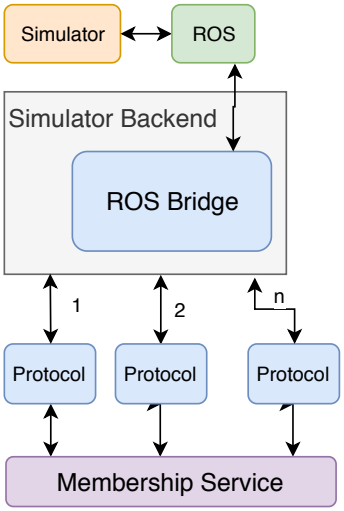



Figure 5.1: Integrated Testing Application Architecture

## 5.2 Integrated Testing Application

In this section, we will show how we developed an application integrating both the agent coordination protocol and the membership service, as well as a visualization layer.

### 5.2.1 Architecture

Figure 5.1 shows the architecture of the application integrating both the coordination protocol, the membership service, and a visualization layer. The purpose of this application is to visualize scenarios where several vehicles utilize the coordination protocol and membership service to coordinate maneuvers among themselves.

The simulator is the application that allows the visualization of the scenario developed. The tool selected and criteria for selection is further described in Section 5.2.2. The simulator backend is responsible for gathering information from the simulator, correctly generating instances of the coordination protocol and assigning them to vehicles in the scenario loaded in the simulator. For this goal it uses a module called ROS bridge. A Protocol is an instance of the coordination protocol, which communicates with an agent simulated in the simulator backend.

### 5.2.2 Simulator

Several simulators were evaluated to use within the integrated testing application, namely:

- Gazebo
- RoboDK
- rviz

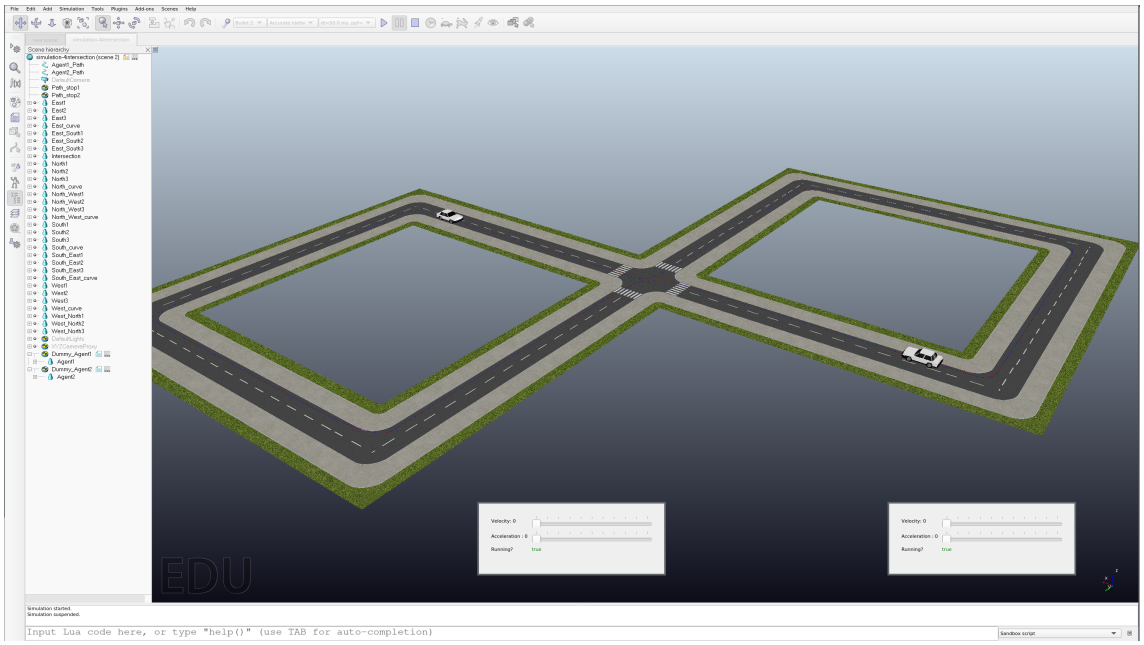


Figure 5.2: V-REP with the scenario developed loaded

- V-REP
- Webots

To select a simulator, a few key points were laid out:

- It should make use of a simple scripting language to program the scenario
- It should provide bindings to interact with ROS

Of the simulators that had both characteristics, Coppelia Robotics platform V-REP[62] was chosen due to making use of the Lua programming language, providing a Lua binding to interact with ROS directly and requiring fewer computing resources to execute.

Figure 5.2 shows the visual representation of the scenario developed for testing the application. It is comprised of a four-way intersection with two vehicles always trying to perform left-hand turns at the intersection. The vehicles loop to return back to the intersection.

Both vehicles execute a lua script, in which they will constantly publish messages in the message facilities provided by ROS, in topic *agentState* with information regarding their position, acceleration and velocity. Additionally, they will subscribe to topic *agentControl*.

When a vehicle is close to the intersection (approximately at the crosswalk) it will publish a message indicating that it intends to execute the tryManeuver procedure, and

stops in its current position. The color of the vehicle will change to yellow. The simulator backend processes this message and invokes the tryManeuver procedure on the corresponding protocol instance, initiating the coordination process.

When each of the remaining vehicles (in this scenario, the only one) grants the requesting vehicle his grant, the simulator backend will publish a message to the *agentControl* topic indicating that the corresponding vehicle should stop. The simulator will stop the vehicle and change its color to red. After the T_GRANT timer expires the vehicle will start moving along its path again.

When the requesting vehicle has received all grants, the simulator backend will inform the simulator by again publishing a message to the *agentControl* topic and the latter will start moving the vehicle again, changing its color to green for the duration of the maneuver. If a grant is not received, the vehicle will keep retrying to get permission.

### 5.2.3 Simulator Backend and Agent Manager

The simulator backend follows a very similar structure to the spawners developed during testing, previously described in Section 4.4.3. It creates a number of simulated agents equal to the number of vehicles in the scenario used in the simulator. An agent manager is also spawned, which acts as a bridge between the simulator backend, and the simulator using ROS' message passing facilities described in Section 4.2.

The agent manager communicates with the simulator by publishing to the *agentControl* topic and subscribing to the *agentState* topic in ROS' publish/subscribe system.

The agent manager also provides its own implementation of the AgentProperties interface, which is based on information received from the simulator. Since the protocol uses just the interface, switching implementations from the spawner provided static implementation to the one provided by agent manager requires no changes to the codebase. It also provides its implementation of the ManeuverCallback interface, that communicates with the simulator to change properties of the vehicles of the simulation whenever the protocol reaches certain conditions.

# Chapter 6

# Evaluation

In this chapter we will evaluate the performance of the coordination protocol under different conditions, as well as reason about the properties that the protocol should guarantee.

## 6.1 Empirical evaluation

To evaluate the safety and liveness properties defined in Section 3.3, we analysed the behavior of the protocol in a scenario with different test cases that comprise most of the expected use cases.

The different cases tested are:

- One vehicle requesting a grant
- One vehicle requesting a grant while a different vehicle holds a grant
- Two vehicles concurrently requesting a grant

These cases enable us to verify that even when creating a situation that directly goes against the desired properties, the coordination protocol can maintain its correct functioning.

The tests were performed using the spawner (see Section 4.4.3). The spawner indicates whenever a grant request is started and obtained. This also enabled us to inject communication failures. Testes were performed for communication failure percentages of 0%, 1%, 2% and 5%.

### 6.1.1 Results

In the first test case, the protocol successfully coordinates the involved vehicles, and the requesting vehicle receives a grant. When facing communication failures, the protocol correctly follows the retry procedure, releasing all vehicles and restarting the process. As such, both safety and liveness properties hold for this test case.

In the second test case, the protocol correctly identifies a maneuver is already taking place, and awaits the termination of that maneuver before requesting one of his own. Similarly, if a new vehicle that was not part of the membership when the current maneuver was negotiated tries to request a grant, existing vehicles will correctly deny his request. Both safety and liveness properties hold for this test case also.

Regarding the third and final test case, the protocol mediates the conflict basing its decision on the timestamp of the requested maneuvers and one of the requests (the one with the earlier timestamp) will prevail, with the vehicle whose request was denied retrying the same request after the accepted one has finished. For this scenario, both safety and liveness properties hold too.

## 6.2 Performance evaluation

In order to evaluate the performance of the protocol in the presence of communication failures, the following metrics were observed:

- Average time to perform a maneuver
- Number of retries per 250 maneuvers

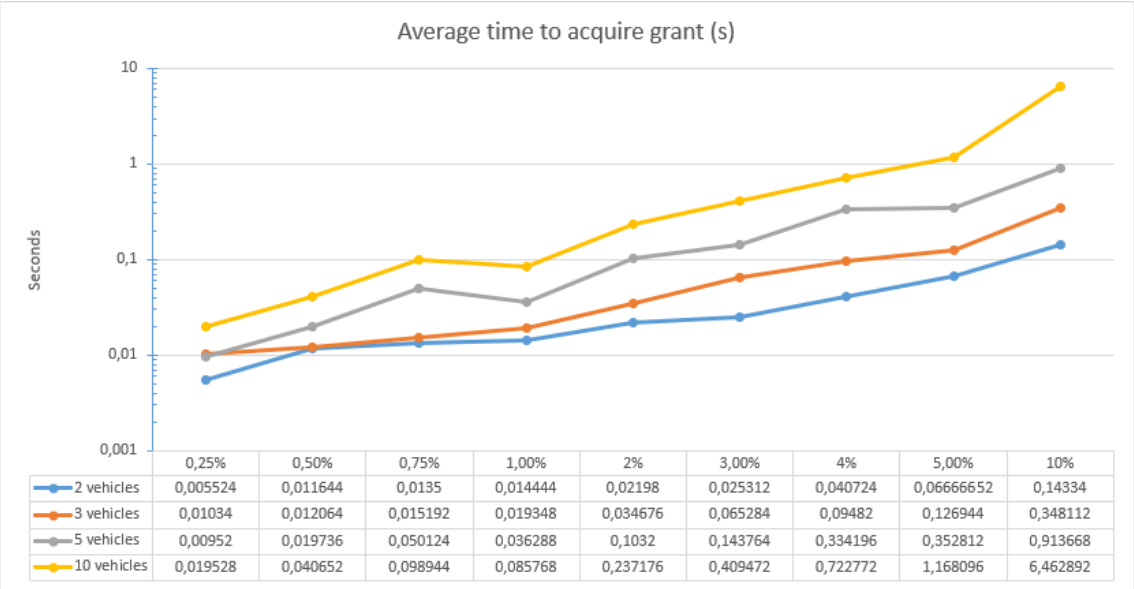
The following parameters were considered to evaluate the latter metrics:

- Communication failure probability
  - $\{0.25\%, 0.5\%, 0.75\%, 1\%, 2\%, 3\%, 4\%, 5\%, 10\%\}$
- Number of vehicles to coordinate
  - $\{2, 3, 5, 10\}$

The communication failure probability values where chosen based on an analysis by Teixeira et al.[64], where it is shown that for distances up to 200 meters average loss rate at 60 km/h is around 1% and up to 300 meters is around 10%. The same analysis also shows that the average delay at 60 km/h is less than 0.1 seconds up to 300 meters. Based on this, a value of 200 milliseconds for $T_D$ was used. The values used for the different timers are in Table 6.1.

| | |
|---|---|
| $T_D$ | 200ms |
| $Ta$ | 1000ms |
| $Tm$ | 300ms |
| $Tman$ | 100ms |

Table 6.1: Parameters used in the coordination protocol evaluation




| | 0,25% | 0,50% | 0,75% | 1,00% | 2% | 3,00% | 4% | 5,00% | 10% |
|---|---|---|---|---|---|---|---|---|---|
| 2 vehicles | 0,005524 | 0,011644 | 0,0135 | 0,014444 | 0,02198 | 0,025312 | 0,040724 | 0,06666652 | 0,14334 |
| 3 vehicles | 0,01034 | 0,012064 | 0,015192 | 0,019348 | 0,034676 | 0,065284 | 0,09482 | 0,126944 | 0,348112 |
| 5 vehicles | 0,00952 | 0,019736 | 0,050124 | 0,036288 | 0,1032 | 0,143764 | 0,334196 | 0,352812 | 0,913668 |
| 10 vehicles | 0,019528 | 0,040652 | 0,098944 | 0,085768 | 0,237176 | 0,409472 | 0,722772 | 1,168096 | 6,462892 |

Figure 6.1: Average time to acquire grant

The spawner (see Section 4.4.3) was used for performing the tests. As such, these tests neglect network latency in their results. However, as mentioned previously, this latency was shown to be under 100 milliseconds for distances up to 300 meters by Teixeira et al. [64].

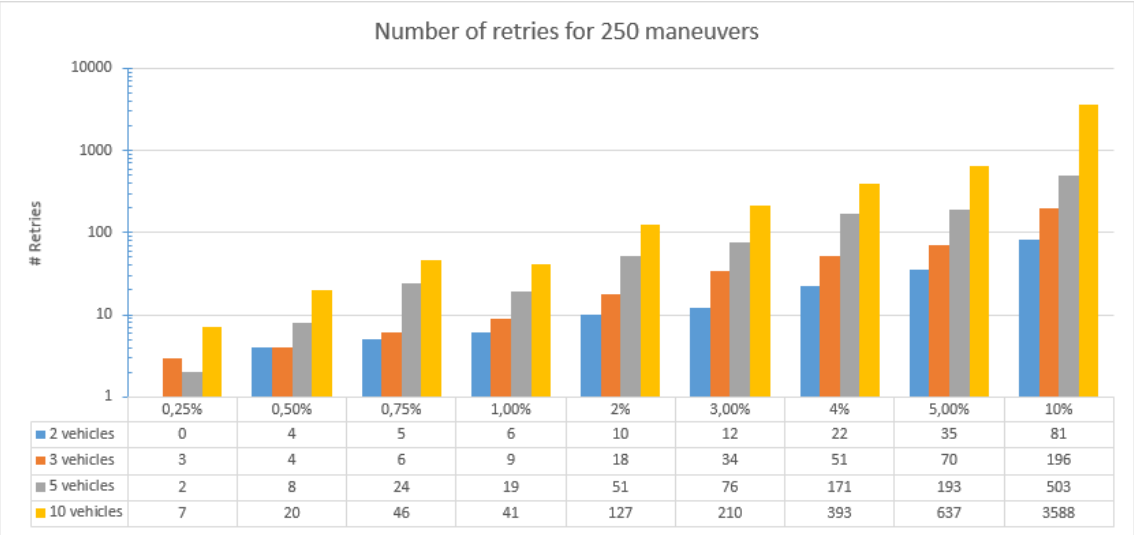
For each combination of values, five runs were performed where a randomly selected vehicle from the membership tried to perform a maneuver. In each run, 50 maneuvers were performed, for a total of 250 maneuvers per combination of values.

### 6.2.1 Results

The results are presented in Figures 6.1 and 6.2.

Regarding Figure 6.1, for 32 of the 36 cases evaluated, the average time to acquire a grant to perform a maneuver was less than 0.5 s. Of the 4 that did not average under 0.5 s, 2 managed to do so under 1 s still. Even then, one of the two remaining cases averaged only just over one second (1.168 seconds). Only the case of 10 vehicles with 10% communication failures deviates from these values with an average of almost 6.5 s.

Hetrick[41] studied the distribution of lane change times using human drivers, which ranged from 3.41 seconds to 13.62 seconds. These times account with the time taken to perform the maneuver itself. To compare the results obtained by Hetrick with the ones obtained in the tests performed, we need to take into account the time a maneuver would take to be performed. A. Goswami[37] developed a trajectory generation algorithm for lane changes in autonomous vehicles, and showed that the vehicles could perform the maneuver in around 5 seconds. Chandra et al.[18] developed an approach to model the critical zone in a lane change maneuver with autonomous vehicles. In their evaluation,




| | 0,25% | 0,50% | 0,75% | 1,00% | 2% | 3,00% | 4% | 5,00% | 10% |
|---|---|---|---|---|---|---|---|---|---|
| 2 vehicles | 0 | 4 | 5 | 6 | 10 | 12 | 22 | 35 | 81 |
| 3 vehicles | 3 | 4 | 6 | 9 | 18 | 34 | 51 | 70 | 196 |
| 5 vehicles | 2 | 8 | 24 | 19 | 51 | 76 | 171 | 193 | 503 |
| 10 vehicles | 7 | 20 | 46 | 41 | 127 | 210 | 393 | 637 | 3588 |

Figure 6.2: Number of retries needed to perform 250 maneuvers

their maneuvers also lasted around 5 seconds.

Considering an additional 5 s on top of our results, the worst case observed in evaluation is still within the range of average lane change times obtained by Hetrick with the highest value being 11,46 seconds. Additionally, the coordination protocol can be executed in a parallel manner, which means that a vehicle can start performing a maneuver with degraded performance to guarantee safety, and when the protocol terminates and the vehicles has a grant, increase performance.

In Figure 6.2 it is shown that the number of retries needed to perform 250 maneuvers grows exponentially as the percentage of communication failures increases. It is also visible that it grows more rapidly the higher the number of vehicles that are part of the membership. As previously stated in Section 3.5, each retry implies an increase in the time needed to acquire a grant of at least $2T_D$. In summary, the values obtained during evaluation are aligned with the values for the average time to acquire a grant.

# Chapter 7

# Conclusion and Future Work

In this chapter conclusions of the work presented will be drawn as well as future work.

## 7.1 Conclusion

This thesis presents an implementation of a coordination protocol for safe cooperation of autonomous vehicles initially proposed in [16], its integration with a cloud based membership service, a testing application for both the coordination protocol and the membership service, and an evaluation of the protocol on a common scenario.

In Chapter 2 an overview of the state of the art was presented. This chapter focused on the current state of autonomous vehicles and existing vehicular applications enabled by V2V communication, as well as technologies to enable that communication.

Chapter 3 provides a detailed overview of the proposed coordination protocol, giving insights on desired properties and presenting the system model used as well as the design of the protocol.

The implementation of the developed coordination protocol is thoroughly described in Chapter 4, as well as the different modules that compose the implementation work together.

Chapter 5 describes the integration of the coordination protocol with the membership service, the communication between both and the testing application developed to test both systems working integrated to enable simulated vehicles to perform a maneuver on a simulated scenario.

Finally, Chapter 6 presents the evaluation carried out to test the implementation of the protocol. Empirical tests were performed to verify that the desired properties are guaranteed in specific test cases. Additionally, performance tests were conducted to evaluate how the protocol reacts in the presence of communication failures. These tests showed that the protocol performs better oor in a comparable manner when compared to human drivers.

## 7.2 Future Work

Regarding future work, one of the items we plan to devise is more scenarios for the testing application integrating the coordination protocol and the membership service. This would allow the visualization of different maneuvers, with different number of cars to assess how the system works under different scenarios.

Another future task would be to implement a testbed of the complete system. This would allow testing the coordination protocol working with the membership service in a physical environment, as well as deploying the system in a realistic architecture, instead of using simulated agents and environments.

Finally, re-evaluating the retry process could improve the performance of the coordination protocol. As demonstrated during the evaluation, communication failures considerably increase the time needed to acquire a grant. If this process could be made more efficient, that is, without compromising the properties of the coordination protocol, it could prove to be highly beneficial. One option worth exploring is not immediately releasing the grants obtained when a communication failure is detected. To avoid compromising other maneuvers, whenever a precedent maneuver is detected, the partial grants acquired could be released, to allow other vehicles to perform their maneuvers.

# Appendix A

# Coordination Protocol - Pseudocode

**Algorithm 1** Agent - Structures and variables

1: **Structures:**
2: $AgentState = (ta, x, v, a)$; ▷ timestamp, position, velocity, acceleration
3: $Agent = (aID, AS)$; ▷ agent ID, AgentState structure
4: $AgentTag = (ts, aID)$; ▷ timestamp, agent ID
5: $Membership = (tm, MO, SM)$; ▷ timestamp, MO flag, SM set of agent IDs
6: $Message = (type, (Agent, AgentTag))$; ▷ message type, message data including Agent and AgentTag structures

7: **Variables:**
8: $AR$; ▷ Agent registry
9: $MR$; ▷ Membership registry
10: $ts$; ▷ Starting time of a maneuver tentative
11: $status$; ▷ NORMAL, GET, GRANT, TRYGET, GRANTGET, EXECUTE
12: $tag$; ▷ Unique tag for each maneuver
13: $grantID$; ▷ ID of the agent with an active GRANT
14: $M$; ▷ Set of received messages
15: $D$; ▷ Set of destination agent IDs
16: $R$; ▷ Set of expected response agent IDs
17: $T\_UPDATE$ ▷ Timer for periodic AR update on storage service
18: $T\_RETRY$ ▷ Timer for retrying to get permission for maneuver
19: $T\_GRANT$ ▷ Timer for the duration of a provided lease

**Algorithm 2** Agent - Interfaces

1: **Interfaces:**
2: $aID()$: get agent ID;
3: $clock()$: read current time;
4: $position()$: read current agent position;
5: $velocity()$: read current agent velocity;
6: $acceleration()$: read current agent acceleration;
7: $createTimer()$: create a new timer;
8: $startTimer(timer, delay)$: start named timer for delay time units;
9: $stopTimer(timer)$: stop named timer;
10: $storeAR(AR)$: write in the storage service an $AR$ registry;
11: $getMR(aID)$: read from the storage service the $MR$ for agent $aID$;
12: $multicast(m, D)$: multicast message $m$ to one-hop distance nodes in $D$;
13: $send(m, d)$: send message $m$ to one-hop distance node $d$;
14: $doManeuver(t)$: execute the maneuver until time instant $t$;
15: $noConflict(AR, t)$: check if conflicts with agent specified in $AR$ can be avoided until $t$;
16: $last()$: returns TRUE if the set $R$ of expected responses is empty, or FALSE otherwise;
17: $granted(aID)$: returns TRUE if $aID$ is the same as $grantID$, or FALSE otherwise;
18: $precedes(rectag)$: returns TRUE if $rectag$ precedes current maneuver $tag$, or FALSE otherwise;

**Algorithm 3** Agent - Initialization and Timer handling.

1: **Initialization:**
2: $(T\_UPDATE, T\_RETRY, T\_GRANT) \leftarrow$
3: $(createTimer(), createTimer(), createTimer())$;
4: $startTimer(T\_UPDATE, TA)$;
5: $(status, grantID) \leftarrow (NORMAL, \perp)$;

6: **upon** timer $T\_UPDATE$ expires, do
7: **begin**
8: $(AR.aID, AR.AS) \leftarrow$
9: $(aID(), (clock(), position(), velocity(), acceleration()))$;
10: $storeAR(AR)$;
11: $startTimer(T\_UPDATE, TA)$;
12: **end**

13: **upon** timer $T\_RETRY$ expires, do
14: **begin**
15: **if** $status = GET$ **then**
16: $status \leftarrow TRYGET$;
17: $multicast((clock(), RELEASE, (AR, tag)), D)$;
18: $tryManeuver()$;
19: **end**

20: **upon** timer $T\_GRANT$ expires, do
21: **begin**
22: **if** $status = GRANT$ **then**
23: $(status, grantID) \leftarrow (NORMAL, \perp)$;
24: **if** $status = GRANTGET$ **then**
25: $(status, grantID) \leftarrow (NORMAL, \perp)$;
26: $tryManeuver()$;
27: **end**

**Algorithm 4** Agent - tryManeuver.

1: **upon** $tryManeuver()$ is invoked, do
2: **begin**
3: **if** $status = NORMAL \vee status = GRANT$ **then**
4: $tag \leftarrow (clock(), aID())$;
5: **if** $status = NORMAL \vee status = TRYGET$ **then**
6: $(status, ts, AR.aID) \leftarrow (TRYGET, clock(), aID())$;
7: $AR.AS \leftarrow (ts, position(), velocity(), acceleration())$;
8: $MR \leftarrow getMR(aID())$;
9: **if** $AR.AS.ta < MR.tm + 2T_M \wedge MR.MO = TRUE)$ **then**
10: $(M, D, R) \leftarrow (\perp, MR.SM, MR.SM)$;
11: **if** $last() = TRUE$ **then**
12: $status \leftarrow EXECUTE$;
13: $doManeuver(clock() + T_{MAN})$;
14: $status \leftarrow NORMAL$;
15: **else**
16: $status \leftarrow GET$;
17: $multicast((ts, GET, (AR, tag)), D)$;
18: $startTimer(T\_RETRY, 2T_D)$;
19: **else**
20: $startTimer(T\_RETRY, TA)$;
21: **else if** $status = GRANT$ **then**
22: $status \leftarrow GRANTGET$;
23: **end**

**Algorithm 5** Agent - Message processing Pt.1

1: **upon** message $m$ received at time $tr$, do
2: **begin**
3: **if** $tr - m.t \leq T_D$ **then** ▷ Message is timely
▷ Grant request response
4: **if** $(m.type = GRANT \vee m.type = DENY) \wedge status = GET$ **then**
5: $(M, R \leftarrow (M \cup m, R \setminus m.AR.aID)$;
6: **if** $last() = TRUE$ **then** ▷ All expected answers received
7: $stopTimer(T\_RETRY)$; ▷ No need for a retry
8: **let** $allgrant = TRUE$; ▷ Reset variable to TRUE before cycle
9: **for all** $minM$ **do**
10: **if** $m.type = DENY$ **then**
11: $allgrant \leftarrow FALSE$; **end if**
12: **if** $allgrant = TRUE$ **then** ▷ Good, no DENYs received
13: $status \leftarrow EXECUTE$;
14: $doManeuver(clock() + T_{MAN})$;
15: $status \leftarrow NORMAL$;
16: **else** ▷ At least one DENY received
17: $status \leftarrow TRYGET$;
18: $multicast((clock(), RELEASE, (AR, tag)), D)$;
19: $startTimer(T\_RETRY, TA)$;
▷ Grant request
20: **else if** $m.type = GET$ **then**
21: **if** $noConflict(m.AR, m.AR.AS.t + 2T_D + T_{MAN}) = TRUE \wedge (status = NORMAL \vee status = TRYGET \vee (status = GET \wedge precedes(m.tag) = TRUE) \vee ((status = GRANT \vee status = GRANTGET) \wedge granted(m.AR.aID) = TRUE))$ **then**
22: **if** $status = NORMAL$ **then**
23: $(status, grantID) \leftarrow (GRANT, m.AR.aID)$;
24: **else if** $status = GET \vee status = TRYGET$ **then**
25: $stopTimer(T\_RETRY)$;
26: **if** $status = GET$ **then**
27: $multicast((clock(), RELEASE, (AR, tag)), D)$; **end if**
28: $(status, grantID) \leftarrow (GRANTGET, m.AR.aID)$;
29: $send((tr, GRANT, \perp), m.AR.aID)$;
30: $startTimer(T\_GRANT, m.AR.AS.t + 2T_D + T_{MAN} - clock())$;
31: **else**
32: $send((tr, DENY, \perp), m.AR.aID)$;

**Algorithm 6** Agent - Message processing Pt.2

▷ Grant release

33: **else if** $m.type = RELEASE \wedge (status = GRANT \vee status = GRANTGET) \wedge granted(m.AR.aID) = TRUE$ **then**
34: $stopTimer(T\_GRANT)$; ▷ No need waiting for end of lease
35: $grantID \leftarrow \perp$;
36: **if** $status = GRANT$ **then**
37: $status \leftarrow NORMAL$; **end if**
38: **if** $status = GRANTGET$ **then**
39: $status \leftarrow TRYGET$;
40: $tryManeuver()$;
41: **end**

# Bibliography


[1] Ieee standard for low-rate wireless networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pages 1–709, April 2016.

[2] Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Technical report, SAE International, 2018.

[3] O. A. Abaza and Z. S. Hussein. Comparative analysis of multilane roundabout capacity "case study". In *2009 IEEE 70th Vehicular Technology Conference Fall*, pages 1–5, Sept 2009.

[4] Mehran Abolhasan, Tadeusz Wysocki, and Eryk Dutkiewicz. A review of routing protocols for mobile ad hoc networks. *Ad Hoc Networks*, 2(1):1 – 22, 2004.

[5] United States Environmental Protection Agency. Sources of greenhouse gas emissions. `https://www.epa.gov/ghgemissions/sources-greenhouse-gas-emissions`. [Online; accessed 30-November-2018].

[6] Marcos K. Aguilera, Gérard Le Lann, and Sam Toueg. On the impact of fast failure detectors on real-time fault-tolerant systems. In Dahlia Malkhi, editor, *Distributed Computing*, pages 354–369, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[7] A. A. Alam, A. Gattami, and K. H. Johansson. An experimental study on the fuel reduction potential of heavy duty vehicle platooning. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 306–311, Sept 2010.

[8] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[9] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: a communication subsystem for high availability. In *[1992] Digest of Papers. FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing*, pages 76–84, July 1992.

[10] O. Babaoglu, R. Davoli, and A. Montresor. Group communication in partitionable systems: specification and algorithms. *IEEE Transactions on Software Engineering*, 27(4):308–336, April 2001.

[11] Y. Bai, K. Xue, and X. Yang. Block mechanism of left-turned flow at signal-controlled roundabout. In *2009 WRI Global Congress on Intelligent Systems*, volume 3, pages 443–449, May 2009.

[12] D. Bauso, L. Giarré, and R. Pesenti. Non-linear protocols for optimal distributed consensus in networks of dynamic agents. *Systems and Control Letters*, 55(11):918 – 928, 2006.

[13] C. Bergenheim, S. Shladover, and E. Coelingh. Overview of platooning systems. In *Proceedings of the 19th ITS World Congress*, October 2012.

[14] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. *SIGOPS Oper. Syst. Rev.*, 21(5):123–138, November 1987.

[15] S. Biswas, R. Tatchikou, and F. Dion. Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety. *IEEE Communications Magazine*, 44(1):74–82, Jan 2006.

[16] António Casimiro and Elad M. Schiller. Membership-based maneuver negotiation in safety-critical vehicular systems. In *Technical report*, Chalmers University of Technology, Department of Computer Science and Engineering, March 2018.

[17] Pew Research Center. Car, bike or motorcycle? depends on where you live. `https://www.pewresearch.org/fact-tank/2015/04/16/car-bike-or-motorcycle-depends-on-where-you-live/`. [Online; accessed 30-November-2018].

[18] R. Chandra, Y. Selvaraj, M. Brännström, R. Kianfar, and N. Murgovski. Safe autonomous lane changes in dense traffic. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, Oct 2017.

[19] Imrich Chlamtac, Marco Conti, and Jennifer J.-N. Liu. Mobile ad hoc networking: imperatives and challenges. *Ad Hoc Networks*, 1(1):13 – 64, 2003.

[20] Gregory V. Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: A comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, December 2001.

[21] Volvo Car Corporation. Drive me - the self driving car in action. `https://www.volvocars.com/intl/buy/explore/intellisafe/autonomous-driving/drive-me`. [Online; accessed 30-November-2018].

[22] Ingemar J. Cox and Gordon T. Wilfong. *Autonomous Robot Vehicles*. Springer, 1990.

[23] M. di Bernardo, A. Salvi, and S. Santini. Distributed consensus strategy for platooning of vehicles in the presence of time-varying heterogeneous communication delays. *IEEE Transactions on Intelligent Transportation Systems*, 16(1):102–112, Feb 2015.

[24] Bertrand Ducourthial, Sofiane Khalfallah, and Franck Petit. Best-effort Group Service in Dynamic Networks. *arXiv e-prints*, page arXiv:0810.3836, October 2008.

[25] ASTM E2213 03. Standard specification for telecommunications and information exchange between roadside and vehicle systems — 5-ghz band dedicated short-range communications (dsrc), medium access control (mac), and physical layer (phy) specifications. Technical report, ASTM, 2018.

[26] N. Fathollahnejad, R. Pathan, and J. Karlsson. On the probability of unsafe disagreement in group formation algorithms for vehicular ad hoc networks. In *2015 11th European Dependable Computing Conference (EDCC)*, pages 256–267, Sept 2015.

[27] Alan Fekete, Nancy Lynch, Yishay Mansour, and John Spinelli. The impossibility of implementing reliable communication in the face of crashes. *J. ACM*, 40(5):1087–1107, November 1993.

[28] M. Ferreira and P. M. d'Orey. On the impact of virtual traffic lights on carbon emissions mitigation. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):284–295, March 2012.

[29] M. Ferreira, R. Fernandes, H. Conceição, W. Viriyasitavat, and O. Tonguz. Self-organized traffic control. In *Proceedings of VANET '10*, September 2010.

[30] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In Marek Karpinski, editor, *Foundations of Computation Theory*, pages 127–140, Berlin, Heidelberg, 1983. Springer Berlin Heidelberg.

[31] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distrib. Comput.*, 1(1):26–39, January 1986.

[32] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[33] National Center for Statistics and Analysis. Summary of motor vehicle crashes: 2016 data. *Traffic Safety Facts, Report No. DOT HS 812 580*, Sep 2018.

[34] Open Source Robotics Foundation. Ros.org — powering the world's robots. `https://www.ros.org`. [Online; accessed 1-December-2018].

[35] Federica Garin and Luca Schenato. *A Survey on Distributed Estimation and Control Applications Using Linear Consensus Algorithms*, pages 75–107. Springer London, London, 2010.

[36] Ali J. Ghandour, Marco Di Felice, Hassan Artail, and Luciano Bononi. Dissemination of safety messages in ieee 802.11p/wave vehicular network: Analytical study and protocol enhancements. *Pervasive and Mobile Computing*, 11:3 – 18, 2014.

[37] Ashesh Goswami. Trajectory generation for lane-change maneuver of autonomous vehicles, 2015.

[38] S. Gräfling, P. Mähönen, and J. Riihijärvi. Performance evaluation of ieee 1609 wave and ieee 802.11p for vehicular communications. In *2010 Second International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 344–348, June 2010.

[39] The Guardian. Uber crash shows 'catastrophic failure' of self-driving technology, experts say. `https://www.theguardian.com/technology/2018/mar/22/self-driving-car-uber-death-woman-failure-fatal-crash-arizona`. [Online; accessed 30-November-2018].

[40] J. . Hermant and G. Le Lann. Fast asynchronous uniform consensus in real-time distributed systems. *IEEE Transactions on Computers*, 51(8):931–944, Aug 2002.

[41] Shanon Hetrick. Examination of Driver Lane Change Behavior and the Potential Effectiveness of Warning Onset Rules for Lane Change or "Side" Crash Avoidance Systems. Master's thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, United States of America, 1997.

[42] Tesla Inc. Autopilot — tesla. `https://www.tesla.com/autopilot`. [Online; accessed 3-December-2018].

[43] D. Jiang and L. Delgrossi. Ieee 802.11p: Towards an international standard for wireless access in vehicular environments. In *VTC Spring 2008 - IEEE Vehicular Technology Conference*, pages 2036–2040, May 2008.

[44] D. Jiang, V. Taliwal, A. Meier, W. Holfelder, and R. Herrtwich. Design of 5.9 ghz dsrc-based vehicular safety communication. *IEEE Wireless Communications*, 13(5):36–43, October 2006.

[45] Idit Keidar, Jeremy Sussman, Keith Marzullo, and Danny Dolev. Moshe: A group membership service for wans. *ACM Transactions on Computer Systems*, 20:191–238, 2000.

[46] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (lar) in mobile ad hoc networks. *Wireless Networks*, 6(4):307–321, Sep 2000.

[47] Yunxin Li. An overview of the dsrc/wave technology. In *QSHINE*, 2010.

[48] Léon LIM and Denis Conan. Partitionable group membership for mobile ad hoc networks. *Journal of Parallel and Distributed Computing*, 74, 08 2014.

[49] Argo AI LLC. Argo ai. `https://www.argo.ai/`. [Online; accessed 30-November-2018].

[50] Waymo LLC. Waymo. `https://waymo.com/`. [Online; accessed 30-November-2018].

[51] Nancy A. Lynch. *Distributed algorithms*. Kaufmann, 1996.

[52] F. J. Martinez, C. Toh, J. Cano, C. T. Calafate, and P. Manzoni. Emergency services in future intelligent transportation systems based on vehicular communication networks. *IEEE Intelligent Transportation Systems Magazine*, 2(2):6–20, Summer 2010.

[53] L. Moreau. Stability of continuous-time distributed consensus algorithms. In *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, volume 4, pages 3998–4003 Vol.4, Dec 2004.

[54] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, C. A. Lingley-Papadopoulos, and T. P. Archambault. The totem system. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers*, pages 61–66, June 1995.

[55] United States Department of Transportation Federal Highway Administration. Intersection safety — fhwa. `https://highways.dot.gov/research-programs/safety/intersection-safety`. [Online; accessed 3-December-2018].

[56] Joshué Pérez Rastelli, Vicente Milanés, Teresa De Pedro, and Ljubo Vlacic. Autonomous driving manoeuvres in urban road traffic environment: a study on roundabouts. In *Proceedings of the 18th World Congress The International Federation of Automatic Control*, Milan, Italy, August 2011.

[57] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, Feb 1999.

[58] H. Qian, K. Li, and J. Sun. The development and enlightenment of signalized roundabout. In *2008 International Conference on Intelligent Computation Technology and Automation (ICICTA)*, volume 2, pages 538–542, Oct 2008.

[59] Joshué Pérez Rastelli and Matilde Santos Peñas. Fuzzy logic steering control of autonomous vehicles inside roundabouts. *Applied Soft Computing*, 35:662 – 669, 2015.

[60] Wei Ren. Multi-vehicle consensus with a time-varying reference state. *Systems and Control Letters*, 56(7):474 – 483, 2007.

[61] Wei Ren, R. W. Beard, and E. M. Atkins. A survey of consensus problems in multi-agent coordination. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 1859–1864 vol. 3, June 2005.

[62] Coppelia Robotics. Coppelia robotics v-rep. `http://www.coppeliarobotics.com/`. [Online; accessed 3-July-2019].

[63] M. Slot and V. Cahill. A reliable membership service for vehicular safety applications. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 1163–1169, June 2011.

[64] Fernando A. Teixeira, Vinicius F. e Silva, Jesse L. Leoni, Daniel F. Macedo, and José M.S. Nogueira. Vehicular networks using the ieee 802.11p standard: An experimental analysis. *Vehicular Communications*, 1(2):91 – 96, 2014.

[65] USA Today. La la land has the world's worst traffic congestion. `https://eu.usatoday.com/story/money/2017/02/20/los-angeles-new-york-and-san-francisco-most-congested-us-cities/98133702/`. [Online; accessed 30-November-2018].

[66] R. A. Uzcategui, A. J. De Sucre, and G. Acosta-Marum. Wave: A tutorial. *IEEE Communications Magazine*, 47(5):126–133, May 2009.

[67] L. Wischhof, A. Ebner, and H. Rohling. Information dissemination in self-organizing intervehicle networks. *IEEE Transactions on Intelligent Transportation Systems*, 6(1):90–101, March 2005.

[68] Qing Xu, R. Segupta, D. Jiang, and D. Chrysler. Design and analysis of highway safety communication protocol in 5.9 ghz dedicated short range communication spectrum. In *The 57th IEEE Semiannual Vehicular Technology Conference, 2003. VTC 2003-Spring.*, volume 4, pages 2451–2455 vol.4, April 2003.

[69] Linjun Yu and Chao Qin. Real-time signal control method for multi-approach roundabouts. 09 2009.

[70] S. Zeadally, R. Hunt, YS. Chen, et al. Vehicular ad hoc networks(vanets): Status, results, and challenges. *Telecommunication Systems*, 50(5):217–241, Aug 2012.