NTNU
Norwegian University of
Science and Technology

Developing a testbed for Intelligent Transportation Systems

Ole Andreas Hansen

Master of Science in Communication Technology
Submission date: June 2018
Supervisor: Wantanee Viriyasitavat, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title: Development of an Intelligent Transportation System Testbed

Student: Ole Andreas Hansen

Problem description: Intelligent Transportation System is the application of sensing, analyzing, control and communication technologies to ground transportation in order to improve safety, mobility and efficiency. ITS will be a central part of the next paradigm within transportation: self-driving cars. While the field is heavily researched and applications are starting to take form there is a gap within the field of testing these types of applications. Current solutions for testing is either performed on a computer or real cars with their own pros and cons.

In this project the student will create a testbed that incorporates the best of computer simulation and testing on real cars. By expanding a previous testbed implementation focus will lie on localization, car movement and a distributed protocol for communications. By using DiddyBorg robots the student will create a multi-purpose platform for testing ITS applications. The implementation will be tested by implementing Virtual Traffic Light algorithm to demonstrate the feasability and effectiveness of Virtual Traffic Lights and the testbed itself.

Responsible professor: Wantanee Viriyasitavat, IIK

Supervisor: Wantanee Viriyasitavat, IIK

Abstract

It is estimated that the average UK commuter spent on average 32 hours stuck in traffic jams in 2017 and that our car park is growing from 1.2 billion cars to 2 billion cars within 2035. In recent years there has been growing research interest in the area of traffic optimization to tackle current and future problems concerning congestion. Intelligent Transportation Systems (ITS) aims to provide innovative services to different modes of transport and traffic management. With the partial introduction of self-driving cars and technology advancements within communication technologies, we can create smarter solutions for traffic optimization and increase traffic flow.

While the field of ITS applications is extensively tested, regarding technical and business perspectives, there are few testbed solutions available to test such applications at a general level. The work performed in this thesis focuses on creating a low-cost testbed by using DiddyBorg robots equipped with sensors to emulate a real-world scenario. The proposed testbed is a hybrid between computer simulations and field tests, enabling researchers and developers to test new ITS applications faster with real-world capabilities. The system is a highly modular framework where new applications can be implemented and tested quickly. To evaluate the testbed, two different applications are developed: regular traffic light and virtual traffic light. Virtual traffic light aims to remove the physical traffic lights by enabling cars to communicate and resolve possible conflicts at intersections. Results show that regular traffic lights, in low-density scenarios, reduce the average waiting time in an intersection by around 90%. Observations and results collected during experiments show that the proposed testbed can give valuable insights to researchers and developers within the field of ITS.

Sammendrag

Det er estimert at den gjenomsnittlige britiske pendleren tilbringer i gjennomsnitt 32 timer i trafikkort hvert år. Med anslagsvis 1,2 milliarder biler i 2014 og en forventet økning til 2 milliarder i 2035, er trafikk optimalisering et viktig forskningsområde. Intelligente Transportsystemer (ITS) er et forskningsområde som tilbyr innovative løsninger og tjenester til ulike transportformer og trafikkstyring. Introduksjonen av selvkjørende biler og teknologiske fremskritt innen kommunikasjonsteknologi gjør plass til nye og smartere løsninger innen trafikkoptimalisering.

ITS-applikasjoner er under omfattende forskning, med hensyn til tekniske og forretningsmessige perspektiver, finnes det få rammeverk for å effektivt teste slike løsninger på et generelt nivå. Arbeidet i denne oppgaven fokuserer på å skape en platform ved bruk av DiddyBorg robotbiler utstyrt med sensorer for å etterligne et virkelighetsnært senario. Det foreslåtte rammeverket er en hybrid mellom datasimuleringer og felttester slik at forskere og utviklere kan teste nye ITS-applikasjoner raskere med elementer fra den virkelige verden. Systemet er utviklet på en modukær møte hvor nye applikasjoner enkelt kan implementeres og testes. For å evealuere plattformen er det utviklet to eksempelapplikasjoner: vanlige traffiklys og virtuelle trafikklys. Virtuelle trafikklys tar sikte på å fjerne de fysiske traffikklysene ved at bilene sels kan kommunisere og løse eventuelle konflikter i et kryss. Resutater viser at virtuelle traffikklys, i et lav tetthetssenario, kan redusere gjennomsnittelige ventetid i et kryss med rundt 90%. Resultater og observasjoner fra eksperimenter viser at den foreslåtte plattformen kan gi verdifull innsikt til forskere og utviklere innenfor ITS-domenet.

Preface

This thesis is the final work of my Master degree at the Department of Information Security and Communication Technology at Norweigan University of Science and Technology (NTNU) in Trondheim, Norway. The thesis is presented as the final project for the MSc program in Communication Technology with specialization in Networks, Services and Applications.

I would like to thank my professor, Wantanee Viriyasitavat for her support and guidance throughout the pre-project and the master thesis. I would also like to thank NTNUI Badminton, Junior Consulting and all the amazing people that i have met over the years in Trondheim. Finally, i would like to thank my family and friends for their support during the last 5 years.

Contents

List of Figures **ix**

List of Tables **xi**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Problem Formulation . . . 3
1.3 Outline . . . 3

2 Background **5**
2.1 Intelligent Transportation System . . . 5
2.2 Vehicle-to-Vehicle Communication . . . 6
2.3 Vehicular Ad Hoc Networks . . . 7
2.4 Virtual Traffic Light . . . 7
2.5 Testbed . . . 8
2.5.1 Computer Simulation . . . 10
2.5.2 Field Test . . . 10
2.5.3 Hybrid Testbed . . . 11
2.6 Related Work . . . 11

3 Methodology **15**
3.1 Requirements . . . 15
3.1.1 List of requirements . . . 15
3.2 Development Methodology . . . 16
3.3 Hardware Components . . . 18
3.3.1 DiddyBorg V2 . . . 18
3.3.2 Raspberry Pi 3 . . . 18
3.3.3 ThunderBorg . . . 20
3.3.4 Camera . . . 20
3.4 Software Language . . . 20
3.5 Testbed characteristics . . . 21
3.5.1 Semi-autonomous driving . . . 22

3.5.2 Localization . . . 22
3.5.3 Simulate driving behavior . . . 24

4 Implementation **27**
4.1 Architecture . . . 27
4.2 Planner . . . 27
4.3 Environment . . . 30
4.4 Location . . . 31
4.4.1 Virtual Location . . . 31
4.4.2 Lane Detection . . . 31
4.5 Network . . . 32
4.5.1 Network Topology . . . 32
4.5.2 Network Module . . . 33
4.6 Motor Control . . . 36
4.7 Car . . . 36
4.8 Applications . . . 37
4.8.1 Regular Traffic Light . . . 37
4.8.2 Virtual Traffic Light . . . 38

5 Simulation **41**
5.1 Simulation setup . . . 41
5.1.1 Simulation Scenarios . . . 41
5.2 Regular Traffic Light Simulation . . . 43
5.3 Virtual Traffic Light Simulation . . . 45
5.4 Results . . . 45

6 Evaluation, Conclusion and Future Work **49**
6.1 Evaluation . . . 49
6.2 Conclusion . . . 51
6.3 Future Work . . . 52

References **53**

Appendices

A Simulation Results **57**

B Configuration settings **59**

C Available ThunderBorg commands **61**

List of Figures

2.1 Illustration of V2V communication between cars and examples of information exchanged. . . . 7
2.2 Principle of operation of the Virtual Traffic Light procedure. Reprinted from [FFCa+10]. . . . 9
2.3 Different type of testbeds for Intelligent Transportation System (ITS) applications . . . 10
2.4 High level development cycle of ITS applications. . . . 12

3.1 The traditional waterfall development cycle . . . 17
3.2 Closed learning loop . . . 17
3.3 Fully assembled DiddyBorg V2 robot . . . 19
3.4 Raspberry Pi 3 . . . 19
3.5 ThunderBorg . . . 20
3.6 Raspberry Pi Camera V2.1 . . . 21
3.7 Output from the different steps in Lane Detection algorithm . . . 25

4.1 Architecture overview . . . 28
4.2 Finite state machine showing the planning process performed by the Planner. . . . 29
4.3 Robots internal map illustrating roads, intersections and the robots position in the system. . . . 30
4.4 Network configuration. . . . 33
4.5 DHCP configuration. . . . 33
4.6 Ad hoc network with Raspberry Pi 3 devices . . . 34
4.7 Sequence diagram showing the flow of a broadcast message . . . 35
4.8 Sequence diagram showing the flow of a application specific message with reply . . . 35

5.1 Images from testbed in action . . . 42
5.2 Simulation scenario 1 . . . 43
5.3 Simulation scenario 2 . . . 44
5.4 Simulation scenario 3 . . . 44

5.5 Graph displaying the average wait time for a green light in an intersection for all three scenarios. 46
5.6 Graph displaying the average speed during the simulations for all three scenarios. 47

List of Tables

4.1 V2V messages sent during VTL procedure. 39

5.1 List of metrics collected during simulations. 42
5.2 Results from regular traffic light simulations 45
5.3 Results from virtual traffic light simulations 45

A.1 Results collected from all robots from the different simulation scenarios with regular traffic light. 57
A.2 Results collected from all robots from the different simulation scenarios with virtual traffic light. 58

Chapter 1
Introduction

1.1 Motivation

With a reported 1.2 billion cars on our roads in 2014 and an expected increase to 2 billion in 2035[1], new challenges for road infrastructure emerges; especially within the field of traffic optimization. According to a report published by the European Union, the cost of traffic congestion is estimated to reach 1% of the European Union's GDP in 2010. Tackling the problem of traffic congestion can roughly be divided into three areas: a) reduce the number of cars and invest in public transport, b) increase the current road infrastructure footprint and c) increase traffic flow on existing infrastructure. It is less plausible that only one of these areas provides the answer to traffic congestion and that a combination of all three areas are needed to reduce the growing congestion problems around the world. Multiple initiatives for the first two areas are already under development and applied to our society. An exciting new area is traffic optimization, using communication technology as a tool to decentralize traffic management and thereby increase the traffic flow using existing road infrastructure.

ITS are advanced applications that aim to provide innovative services relating to different modes of transport and traffic management. Two important properties of ITS are Vehicular Ad Hoc Network (VANET) and Vehicle-to-Vehicle (V2V) communication between cars. V2V creates temporary ad hoc networks for communication between cars, enabling new services related to different modes of transport and traffic management to be created. ITS services will give cars and drivers the opportunity to be better informed and make safer, more coordinated, and smarter use of transport networks [DD10].

One proposed application ITS is Virtual Traffic Light (VTL), a proposed self-organized traffic control system that utilizes VANET with the goal of reducing traffic congestion [FFCa+10]. Allowing cars to communicate their presence, speed and other

[1] https://www.drivesurfing.com/en/article/7/how-many-cars-are-there-in-the-world

information, can eventually lead to the removal of physical traffic lights and reduce traffic congestion by dynamically optimizing traffic flow based on the current traffic situation.

With VANET and ITS we can start to rethink the way we solve traffic optimization and security. Current solutions to traffic optimization often have a centralized approach. Meaning that they are often complex and unsuitable for adapting, in short timescales, to context changes in the environment. For example, each traffic light is governed by a computer that often uses predefined timing strategies to control an intersection. It does not change according to the time of day and number of vehicles. Allowing cars do communicate enables the opportunity of decentralizing such systems to leverage the potential decentralized mobility systems provide. An excellent example of this is VTL.

According to the Norwegian Public Roads Administration, Norway has around 2400 traffic lights. In a report published by the Institute of Transport Economics [Hø15], it is stated that the installation cost of one traffic light is on average $126,000 with an annual operation cost of $12,600 (10%). Implementing VTL can potentially save $30.5 million a year in operational cost. According to M. Ferreira et al. [FFCa$^+$10] the annual operation cost in the U.S. is $780 million. Potentially, VTL can reduce the amount of time spent in traffic jams while also massively save governments millions of dollars in operating costs.

VTL is one area where ITS shows great potential, and while such systems are under heavy research ([FFCa$^+$10], [Fd12], [NAT$^+$12], [CFS13]) there are few good solutions for testing ITS applications in a real-world scenario. Current solutions are either simulated on a computer or tested on actual cars, both with their pros and cons. Computer simulations provide quick feedback in an early research and development stage but lack the variability that real-world scenarios provide. Performing field tests is highly time and cost consuming and also introduces many security challenges. It is also somewhat unpractical since it requires a large, closed environment to perform tests in a safe environment.

Particularly, this project goal is to create a low-cost testbed to give research and developers quick access to testing their solutions while providing the variability of real-world scenarios. Testing can be performed in a closed environment, on physical robots, and in a realistic scenario. The result of this project aims to provide an environment where the challenges of indoor simulation and initial setup is solved and transparent for the user – reducing the time needed for preparations. To achieve this, research within indoor positioning, autonomous driving, and simulation of real-world driving is needed. Then the current VTL implementation from A. Brastad's thesis [Bra17] is extended to include these objectives.

1.2 Problem Formulation

This thesis presents a proposed architecture and implementation of a low-cost testbed for ITS applications. The main purpose of this research is to address the area of combining computer simulations with real-world characteristics. The following objectives are deducted from the problem description:

- Perform a literature study within the field of ITS
- Combine research within self-driving cars with small-scale robots
- Design a general purpose testbed prototype for testing ITS applications
- Perform simulations on the implemented testbed to evaluate its performance
- Compare and evaluate the proposed implementation against related solutions

The problem statement also contains additional tasks. Research within localization and communication technologies to have indoor localization and a supplement for VANET.

1.3 Outline

The following chapters are structured as follows:

Chapter 2: Presents an overview of technologies within ITS, an introduction to the term testbed, and a comparison between different testbed categories before discussing related works within the field of testing ITS applications.

Chapter 3: Presents the methodology, requirements for the testbed, hardware and software components and some critical components researched during the project.

Chapter 4: Presents the overall system architecture and explains the implementation of each module that is needed for the testbed. It also presents and describes the different applications that are used to evaluate the testbed.

Chapter 5: This chapter introduces three different simulation scenarios that are used during simulations, how simulations were conducted and a discussion about results from the simulations.

Chapter 6: This chapter discusses observations and results from simulations to evaluate the effectiveness of the testbed. We also highlight some problems of the current testbed implementation before concluding and present some comments for further work.

Chapter 2
Background

This chapter explores the area of Intelligent Transportation Systems, its surrounding technologies, the concept of a testbed before ending with an overview of related work and current solutions.

2.1 Intelligent Transportation System

> Intelligent Transport Systems or ITS means systems in which information and communication technologies are applied in the field of road transport, including infrastructure, vehicles, and users, and in traffic management and mobility management, as well as for interfaces with other modes of transportation. ([otEU10])

ITS are advanced applications that aim to provide innovative services relating to different modes of transport and traffic management. An ITS service is the provisioning of an ITS application through a well-defined framework with the aim of contributing to user safety, efficiency and comfort. ITS is a part of Internet of Things (IoT) that includes V2V and Vehicle-to-Infrastructure (V2I) communication. While the ITS branch is broad, this project will mainly focus on the sub-branch Traffic Optimization, where the goal is to reduce the time a vehicle is not driving in road traffic. With the introduction of self-driving cars, ITS will play an essential role to increase efficiency and to help reduce congestion and CO2 emissions. All though self-driving cars are not yet part of our everyday life, ITS applications are already contributing to increase safety and optimize traffic flow. Most modern cars now use a lane departure warning system [MT06], which is a system designed to warn drivers when a vehicle begins to move out of its lane. This system is designed to minimize accidents such as driver error, distractions, and drowsiness. Another example that optimizes traffic flow and increases security is adaptive cruise control [VE03]. This cruise control system automatically adjusts the vehicle speed to maintain a safe

distance from vehicles ahead - making the flow of vehicles smoother and more efficient by always having the optimal speed and distance from the car in front.

It is safe to assume that ITS applications will play an essential role for self-driving cars and the future. Governments and the European Union already have directives in play to ensure standardization and speed up the process of introducing ITS applications[1]. Car manufacturers are starting to implement support for ITS applications in their new models.

2.2 Vehicle-to-Vehicle Communication

V2V is an ad-hoc networking paradigm and a vital part of ITS applications. Vehicular communication systems are networks in which vehicles and roadside units are the communicating nodes, exchanging information with each other such as safety warning, traffic information, and location beacons. The system is capable of using multiple types of communication technologies, and the United States Congress has reserved a region in the 5.9 GHz band for V2V communication, named the Dedicated Short Range Communication (DSRC) band.

The goal of V2V is to prevent accidents by allowing vehicles in transit to send position and speed data to one another over an ad-hoc mesh network. It is also expected to be more effective than current automotive original equipment manufacturer (OEM) embedded systems such as lane departure warning, adaptive cruise control, and blind spot detection. V2V has the ability to have a ubiquitous 360-degree awareness of surrounding threats by constantly receiving information beacons about its surroundings. Figure 2.1 illustrates the principle of V2V communication between cars and different types of information exchanges.

In 2017 the US National Highway Traffic Safety Administration (NHTSA) proposed a rule to require all new vehicles to have V2V capabilities where the primary goal is that by 2023 all new vehicles should be able to communicate with each other[2]. The proposal also includes standardization of messages that vehicles will exchange. During the 90-day comment period, many comments were received, and critical topics addressed include privacy and security, technology strategy, implementation timing and cost estimates. Currently, it is the significant scope of the rule that makes it hard to implement. Similar initiatives are also under development in Europe by the Car-2-Car Communication Consortium[3].

[1] https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX%3A32010L0040
[2] https://www.federalregister.gov/d/2016-31059
[3] https://www.car-2-car.org/

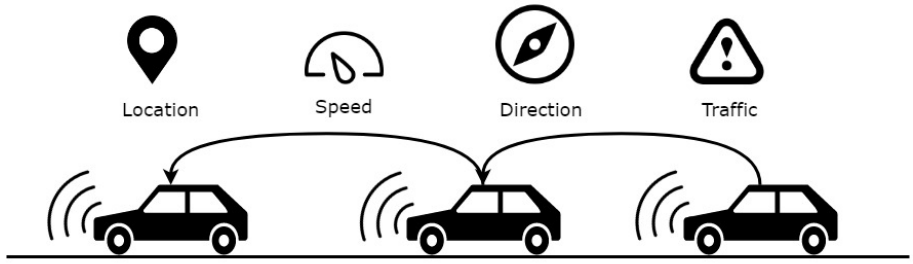


Figure 2.1: Illustration of V2V communication between cars and examples of information exchanged.

2.3 Vehicular Ad Hoc Networks

VANET uses the principles of mobile ad hoc networks which is the creation of a wireless network for data exchange and applying this principle to the vehicular domain. VANET is the framework used to achieve V2V communication and also providing services to V2I communication. Communication is achieved by wireless access technologies such as IEEE 802.11p which adds support for wireless access in vehicular environments to the IEEE 802.11 standard. This amendment and continuous innovation in wireless communication aim to improve road safety and road traffic efficiency by creating a framework in order to develop and support ITS.

In [EZL14] by E. C. Eze et al. a review of VANETs current state, challenges, and potential was conducted. Based on their conclusion, VANET is no longer a remote feasibility. Substantial investments by both governments and car manufacturers are already in the pipeline and under development. E. C. Eze et al. also point out that several unique open research challenges ranging from wireless network evolution, reliable message dissemination to event detection. These unanswered questions make research within the field of VANET attractive. Finally, they point out that besides recent technical development, another critical and important phase for VANET that will drive success is systematic commercial marked introduction and acceptance. Despite the challenges, there is a uniform agreement that VANET will be the framework used to achieve V2V and V2I communication.

2.4 Virtual Traffic Light

VTL in an example of an ITS application that utilizes the properties provided by VANET. VTL is a self-organizing traffic control concept presented in [FFCa$^+$10]. M. Ferreira et al. proposes and present preliminary results on the mitigation of traffic lights as roadside-based infrastructure to in-vehicle virtual sign supported by only V2V communication. The idea is to design a VTL protocol that can dynamically

optimize the flow of traffic in road intersection without requiring any roadside infrastructure. A decentralized approach to a centralized solution. Figure 2.2 shows the principle of operation of VTL. When vehicles approach an intersection, they look for potential conflicts - other cars - within the intersection area. If such a conflict exists, a leader is elected to act as temporary road junction infrastructure and broadcast traffic light messages to surrounding cars. The idea is simple: use information and communication technology to decide which vehicle is presented with a green light in intersections. Notably, for such a solution to work, M. Ferreira et al. present a list of assumptions for a VTL implementation:

- All vehicles are equipped with DSRC devices.
- All vehicles share the same digital roadmap
- All vehicles have global positioning system (GPS) device that guarantees global time and position synchronization with lane-level accuracy.
- The security, reliability, and latency of the wireless communication protocol are assumed to be adequate for the requirements of the VTL protocol.

Interestingly, the VTL algorithm presented in [FFCa$^+$10] lacks several details as pointed out in [BZMP14]. Details such as the definition of messages that need to be exchanged and the rules that are applied to govern the green-to-red cycles. Moreover, only broadcast messages are sent by the junction leader, with no acknowledgments and thus no possibility to check their correct reception. A. Bazzi et al. propose a novel VTL distributed algorithm, where the exchange of information between vehicles occurs using both broadcast messages for signaling and unicast messages for precedence definition and traffic light decisions [BZMP14]. The implementation presented by A. Bazzi et al. will serve as the reference implementation of VTL in this project.

2.5 Testbed

A testbed is a platform for conducting rigorous, transparent and replicable testing of scientific theories, computational tools, and new technologies. The definition of a testbed can be applied to all stages from research and development to completion, where the testbed is created specifically for each phase. The term is used across many disciplines to describe research and new product development platforms and environments. They can vary from hands-on prototyping in manufacturing such as vehicles, aircraft or systems and to intellectual property refinement such as software development. Sufficient testing is an essential part of research and development activities. With the growing attention for ITS applications, different solutions for

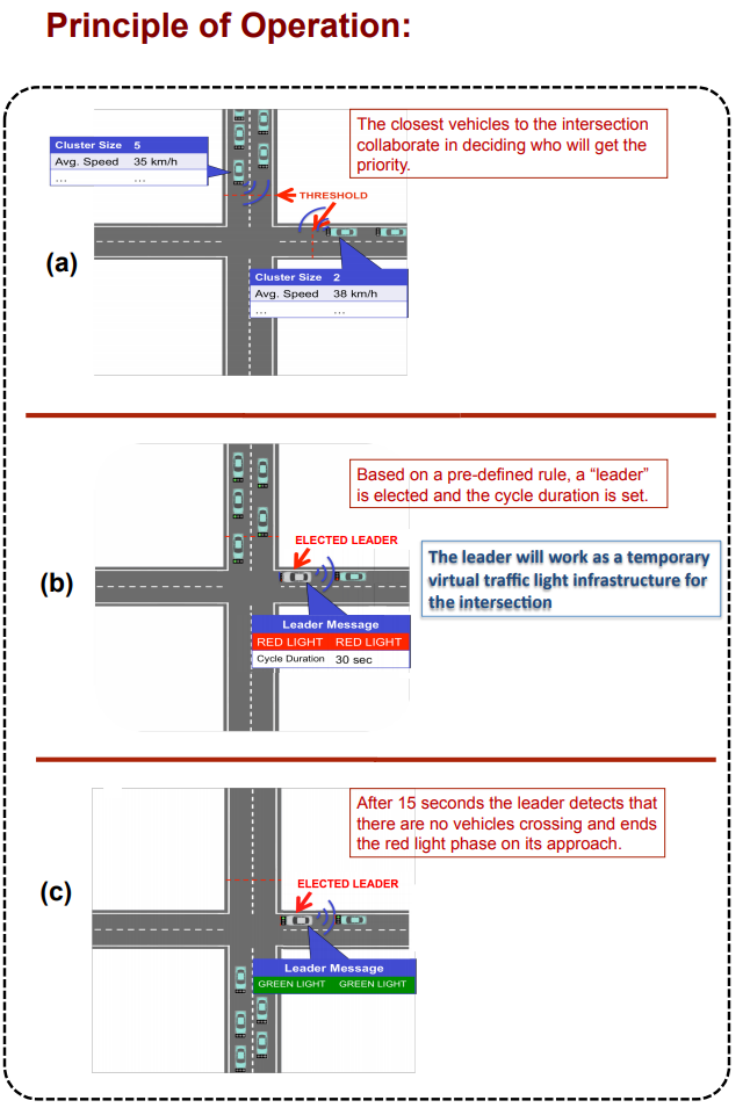
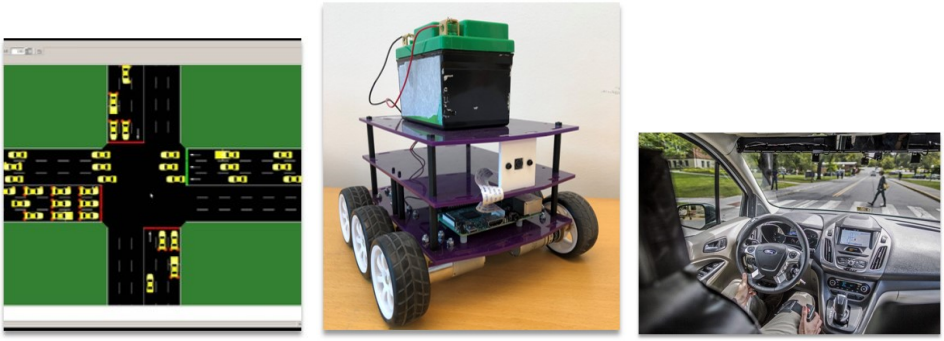


Figure 2.2: Principle of operation of the Virtual Traffic Light procedure. Reprinted from [FFCa+10].



(a) Computer Simulation (b) Hybrid Testbed (c) Field Test

Figure 2.3: Different type of testbeds for ITS applications

ITS testbed exists and can be divided into three categories: computer simulation, field test and a hybrid solution (figure 2.3).

2.5.1 Computer Simulation

Computer simulation (figure 2.3a) consists of simulating the whole environment, structure, and logic on a computer. Such testbeds can yield quick results and can easily be scaled to accommodate thousands of vehicles and large simulation environments. One example is the open source program Simulation of Urban MObility (SUMO). SUMO is a highly portable microscopic and continuous road traffic simulation package designed to handle large road networks [KEBB12]. It comes with an easy to use GUI and the option to import maps from OpenStreetMap to simulate lane-driven traffic. The system can handle thousands of cars with the option to have multiple types of cars with their own attributes. SUMO is a great tool for simulating lane-driven traffic, and when performing proof of concepts and large simulations early in a research and development stage. However, an important factor for ITS success and self-driving cars are sensors and sensor data. Modern cars use a range of sensors to collect and act on data. Many proposed ITS applications will gain an advantage of utilizing sensor data to respond autonomously to events. The main drawback for computer simulations is that they can not provide sensor data and therefore not be a suitable testbed for certain ITS applications.

2.5.2 Field Test

As opposed to Computer Simulations, field testing involves performing tests on real cars in a live environment (figure 2.3c). This type of testing is one of the best ways to test ITS applications since the scenario and environment is closer to the actual world compared to computer simulations. However, such tests also have some limitations.

The biggest challenges of performing physical tests are logistics, time and cost. Tests have to be performed in a closed environment for safety purposes, and such tests often need a regulatory approvement from the government. Which will increase both the cost and time needed to perform tests. Costs associated with equipment, people, and time are also an important factor when planning and performing field tests. Weather can also provide some constraints since a test might depend on different weather conditions. With all this in mind, it is important to point out that there are no ways of completely removing physical testing when developing new ITS applications. In fact, it is the only test that can provide insights into how the system responds in a real-world scenario. To get the most out of such tests, the application should be close to fully implemented and bug-free. Implying that physical tests should be performed when the application is close to completion and can move on to its commercialization phase.

2.5.3 Hybrid Testbed

It is clear that both computer simulation and physical testing have their pros and cons. A hybrid solution (figure 2.3b) tries to fill a gap in the field of testing ITS applications by combining the speed and modularity from computer simulations with sensor-data and real-world scenarios from field testing. Conducting full-scale ITS experiments encounters many challenges as described above; a hybrid solution will shorten the time and footprint needed to conduct experiments. A hybrid solution is not perfect but can give valuable insights and results that computer simulations and physical testing might not cover within time and cost constraints. Advancements in other areas such as robotics, sensors, computers, and open source machine learning algorithms make a hybrid solution easier to create and implement compared to just a few years ago. To better give an illustration of where a hybrid solution would fit in we can divide the development of an ITS application into three parts: research & prototyping, development and ready for market. Figure 2.4 shows where the different testbed solutions give the highest level of information gain when conducting tests.

2.6 Related Work

The field of ITS is big with many research initiatives starting to grow in scale and attention. The range of ITS initiatives includes simple computer simulations to big connected roads pilots being introduced in many countries. In this section, we will introduce related work within the field of ITS applications.

Computer Simulations
M. Ferreira et al. proposes a self-organized traffic control scheme by implementing VTL. The proposed VTL protocol was implemented in the DIVERT simulator [CaDFB08]. A large-scale simulator that allows for microsimulation of thousands of

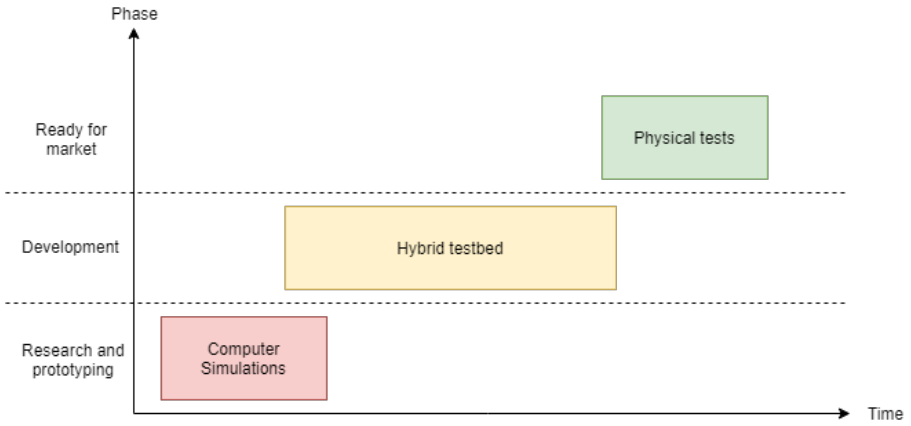


Figure 2.4: High level development cycle of ITS applications.

vehicles with a high degree of realism. Simulations were performed at four different traffic densities. Their results show that the percentage benefit of VTL increases substantially as traffic density becomes higher, resulting in a 60% increase in average traffic flow rates. Showing that the proposed self-organizing traffic paradigm holds the potential to revolutionize traffic control, especially in urban areas. In light of M. Ferreira et al., many papers have studied the effects of the VTL protocol. T. Neudecker et al. studies the VTL protocol under challenging condition such as non-light-of-sight environments [NAT+12]. M. Ferreira and P. M. d'Orey study the impact of VTL on carbon emissions showing that the VTL protocol can reduce carbon emissions up to 18%.

In [ZYY+17], W. Zhou et al. creates a testbed to study intelligent control algorithms within ITS and their effect on vehicle energy consumption. Their testbed uses SUMO and Matlab to create an Intelligent Driving Simulation Platform. When testing different Vehicle-Infrastructure Cooperation algorithms, results show the potential to reduce energy consumption and ease traffic congestion.

Field Test

Field test initiatives are mostly sponsored by government and large automobile companies such as BMW, Volkswagen, and General Motors. In Europe especially, many countries have their own implementation of a field test within ITS applications. Most noticeable is the C-Roads[4] initiative sponsored by the European Union and member states. The C-Roads initiative aims to accelerate testing and implementing ITS services where multiple member states including Austria, France, Germany, and the Netherlands have test-sites available to test ITS applications in a real-

[4]https://www.c-roads.eu/platform.html

world scenario. Another example is Virginia Smart Road in the US. A full-scale, closed testbed research facility consisting of 9.1km roads built with Federal Highway Administration standards. Its main features include connected-vehicle systems, weather systems, visibility testing system and lighting system. Over 18,000 research hours have been logged in the Smart Road and is the only facility in North America that can provide a wide array of test environments in the field of transportation.

Hybrid Testbed

Some hybrid testbed implementations within the field of ITS have been developed the last few years as seen in [BKV+09], [TTB+15], [LLD+12] and [Bra17]. Common features for these implementations is that they are all developed for specific use cases and hard to extend to a more general level. Most notably is the work performed by S. Biddlestone et al. in [BKV+09] where an indoor intelligent transportation testbed for urban traffic scenarios was developed. Indoor localization is one of the hardest challenges to solve when creating such a testbed. Their implementation uses a number of modules to achieve just this, making the localization quite robust. A Virtual GPS designed to use the ARTKPlus Toolkit and two Point Gray Research Firewire Cameras (Scorpion and Bumblebee) to track the robots as they maneuver through the environment. This virtual GPS is then combined with dead reckoning and a virtual LIDAR to find the position of the robot accurately in the environment. Its implementation also supports multiple types of vehicle behavior such as parking and scheduling, intersection behavior and convoying. The testbed implemented by S. Biddlestone et al. has been successfully used in various applications and provided useful insights. One of the drawbacks of this implementation is as mentioned in section 2.5.1, and this is the lack of sensor data from sensors used in modern cars. Which restricts the types of ITS applications that can be tested with this implementation.

In A. Brastads master thesis [Bra17] a simple testbed for VTL was created using DiddyBorg robots programmed to follow a fixed route on a 2D map. The implementation of the VTL algorithm follows the concept described by M. Ferreira et al. in [FFCa+10] and preliminary results shows that VTL is approximately 18% more effective compared to regular traffic lights. This project is a continuation of the work performed by A. Brastad in his thesis. While the work performed by A. Brastad focused around creating a testbed specifically for VTL this project aims to create a more general testbed for multiple types of ITS application. Many of the components specified in [Bra17] serves as the basis for this project as described in chapter 3.

Chapter 3
Methodology

This chapter serves as the foundation for the design and implementation of the testbed. First, we introduce some requirements for the testbed which are the guidelines for the architecture, design, and implementation. We then present and discuss the development methodology used to create the testbed before introducing the different hardware and software components used throughout the project. Lastly, we introduce the main characteristics of the testbed and the process of why they were chosen.

3.1 Requirements

As mentioned in section 2.6.3, this project objective is to create a low-cost hybrid testbed for ITS applications. The initial requirements to achieve this are listed below, serving as the primary guidelines during the design and implementation phase. The requirements are a result of the theory described in chapter 2 and literature research within the different fields.

3.1.1 List of requirements

Each requirement is denoted as **RQx** where x is the requirement number.

RQ1: An ITS testbed should be flexible and support multiple types of environment such as different maps, intersections, and roads.

RQ2: The testbed should be designed in a decentralized manner to support multiple robots. Each robot should be as independent from another as possible

RQ3: A system to emulate VANET must be present for robots to communicate during simulations.

RQ4: Robots should have a dynamic route and not follow a pre-programmed fixed route during simulations.

RQ5: Robots should enforce regular traffic rules and behavior such as driving on the right-hand side, be aware of other cars and adapt speed according to the situation.

RQ6: The robots should have lane-level accuracy during simulations and make adjustments to the robots heading. An indoor GPS supplement is needed.

RQ7: The system should have a low cost and simulations should be easy to perform without having to spend hours of configuring the system.

3.2 Development Methodology

Work performed in this project consists of four primary phases:

- Theory and literature study
- Prototyping and proof-of-concept
- Design and implementation
- Simulation and evaluation

An important factor for this project is that the four phases are partially performed in parallel by applying an agile development methodology. Multiple development methodologies exist and are ever changing. The traditional waterfall development method have in recent years been replaced by a more agile or lean process to accelerate the development processes and reducing time and cost. The traditional waterfall development methodology is a linear and sequential process where the completion of one phase is the beginning of the next. Disadvantages of waterfall model are that no working software is produced until late during the projects life cycle. It is also not a good model for complex and object-oriented projects. A general overview of the waterfall model is shown in figure 3.1. As seen in the figure the process is sequential, and there are no easy ways to go back to a previous step without project delays and increased cost. This is where an agile approach comes in, the idea is simple: make the process iterative and introduce a closed learning loop. At the start of an iteration is learning. We gain knowledge about a field. Based on this knowledge we generate solution ideas and assumptions. These ideas and assumptions are then quickly developed. The next step is to test the solution and see if the idea were realized and the assumptions confirmed if so the product or work is kept. If an idea or assumption does not work as expected, it can be abandoned. Independent of the outcome, new knowledge about the field and the product is gained, and the process is restarted. Figure 3.2 shows the iterative learning cycle based on an agile process. Throughout this project, the iterative, agile approach has been applied to answer the requirements specified above.

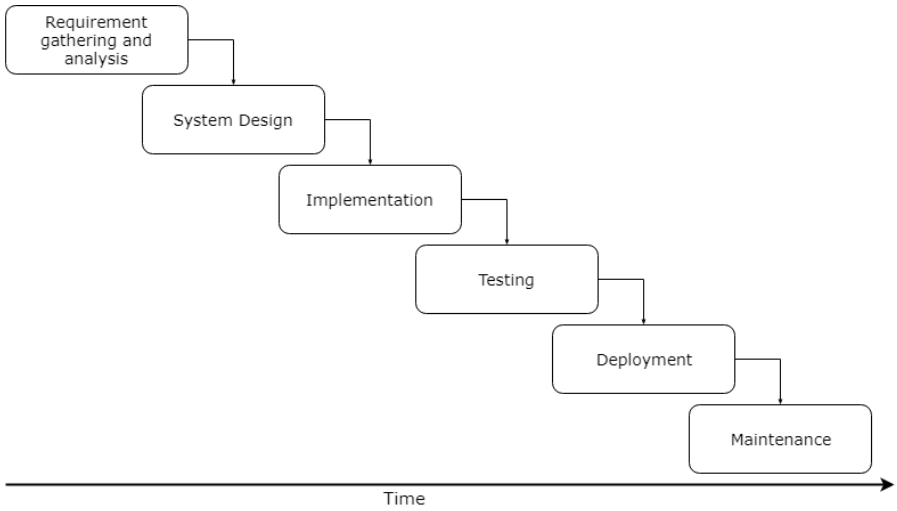


Figure 3.1: The traditional waterfall development cycle

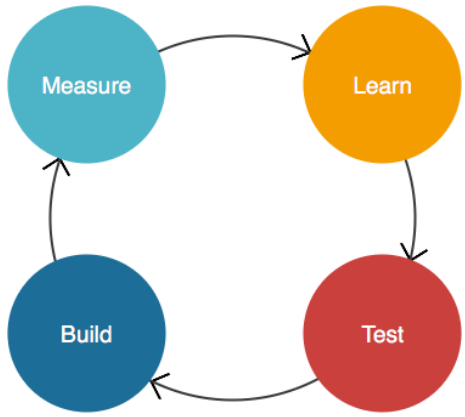


Figure 3.2: Closed learning loop

3.3 Hardware Components

This section introduces the different hardware components and their specifications used to create the testbed.

3.3.1 DiddyBorg V2

For this project, the choice of robots is the DiddyBorg V2 robot which is an upgraded version from the DiddyBorg V1 previously used in A. Brastad's thesis [Bra17]. DiddyBorg V2 is a six-wheeled high-torque robotics platform powered by batteries. It uses six 12V DC gear motors with three mounted on each side of the robot. Its central controlling unit is an onboard Raspberry Pi 3 making it highly customizable and suitable for this project.

The DiddyBorg robots are one of the most powerful robots available on the market and is easily controlled through a Python API provided by the PiBorg Organization. It can drive over most indoor and outdoor terrain, can climb inclines up to 45 degrees and perform a 360-degree turn which makes it suitable for a testbed. Compared to existing solutions for testing ITS application, the simulations can be executed indoor in a controlled environment as opposed to an outdoor test with more variability in its environment. The robots have a low cost - 210 pounds - and are easily customizable and extendable for new use-cases. Figure 3.3 shows one of the assembled DiddyBorg robots.

3.3.2 Raspberry Pi 3

Raspberry Pi 3 (figure 3.4) is a small credit-card-sized computer at a low cost. It can be used as a small personal computer, a media center or as a controlling unit in an electronics project. It comes with a variety of different Operating Systems which is chosen for different types of use. For this thesis, the Raspberry Pi is installed with Raspbian as its Operating System. Raspbian is based on Debian and optimized for the Raspberry Pi hardware. It provides an easy to use GUI which makes development and configuration of the unit more accessible.

Raspberry Pi 3 specifications:

- Quad Core 1.2GHz 64bit CPU
- 1GB RAM
- 4 USB 2 ports
- Full size HDMI port
- 40-pin extended GPIO

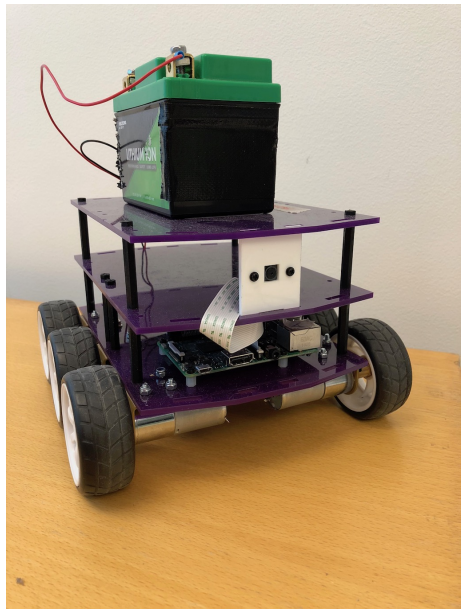


Figure 3.3: Fully assembled DiddyBorg V2 robot

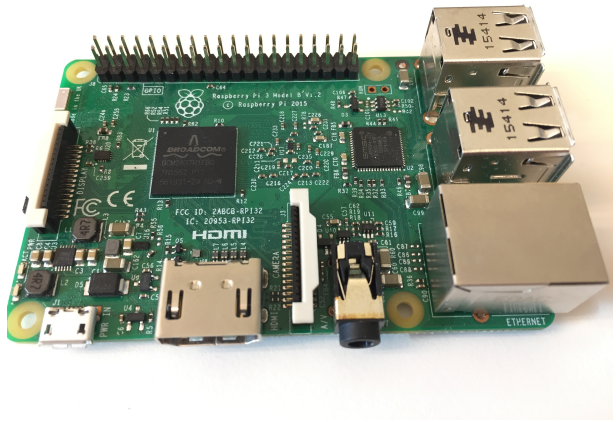


Figure 3.4: Raspberry Pi 3

– Wireless LAN and Bluetooth Low Energy on board

– CSI camera port for Raspberry Pi Camera

3.3.3 ThunderBorg

ThunderBorg is a powerful dual motor control board which makes it possible to power the DiddyBorg motors and the Raspberry Pi with batteries instead of a USB supply. ThunderBorg is directly connected to all six motors and the Raspberry Pi via GPIO. The PiBorg organization also provides an easy to use Python API for controlling the motors. A list of available commands is listed in appendix C. The ThunderBorg board is shown in figure 3.5

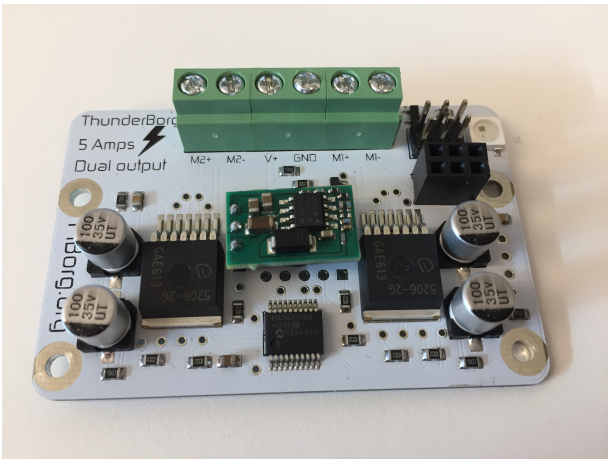


Figure 3.5: ThunderBorg

3.3.4 Camera

For the robots to have self-driving capabilities, they are fitted with a Raspberry Pi Camera (figure 3.6). The camera module can record video up to 1080p at 30 frames per second or 720p at 60 frames per second. With this high definition video, we can use image processing techniques such as lane detection used in this project. The camera is mounted in from of the robot and is used as a solution for indoor localization together with software techniques.

3.4 Software Language

The testbed is implemented using Python as the primary programming language with the support of multiple Python packages. Python was chosen mainly for two reasons; the previous implementation by A. Brastad in [Bra17] and Pythons easy syntax. Python requires less overhead to get started and have a solution implemented and tested. Most modern libraries such as open-cv and numpy used in this project provide a python package for easy implementation. One major drawback of Python that was

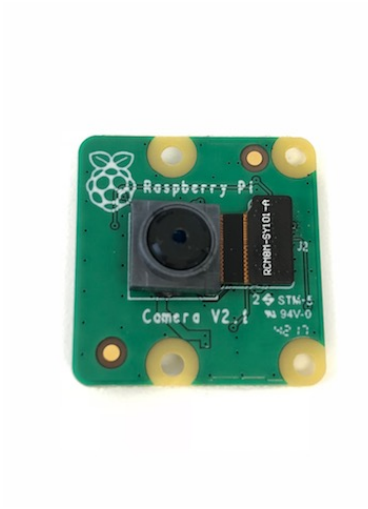


Figure 3.6: Raspberry Pi Camera V2.1

discovered during this project is the Global Interpreter Lock (GIL)[1]. Python offers the option of multithreading, but the GIL can only allow one thread to be executed at a time which means that vi reduce the CPU effectiveness of multithreading. However, this can be solved using an implementation of multiprocessing instead. Instead of creating threads, we create processes, but we also lose some options concerning communication and shared objects between the processes. Many of the modules developed in the testbed are either a thread or a process. All modules started as a thread, but if the program was not effective enough, it was migrated to a process instead. PyCharm Professional Edition[2] (Student Licence) was used as the primary IDE when developing the testbed.

3.5 Testbed characteristics

This section addresses some key characteristics of the testbed and theory behind them. Based on the requirements in section 3.1.1 we can identify some critical areas that the testbed needs to implement. Concerning the testbeds context and domain - ITS - we have identified three main characteristics as the projects primary objective: semi-autonomous driving, localization, and human driving behavior.

[1] https://wiki.python.org/moin/GlobalInterpreterLock
[2] http://www.jetbrains.com/pycharm/

3.5.1 Semi-autonomous driving

RQ1 states that the testbed must support changes in its simulation environment. The best way to achieve this is to give the robots some autonomous driving capabilities. In the previous testbed ([Bra17]), the robots were programmed to follow a fixed route and drive in a loop. The proposal for this project is to create functionality where the robots take an independent decision of where to drive. Making the robots drive a dynamic and different route based on its decisions. We assume that this approach will be closer to a real-world scenario. As **RQ5** states: robots should enforce regular traffic rules and behavior such as driving on the right-hand side, be aware of other cars and adapt speed according to the situation. To support this, multiple mechanisms such as map exploration, out-of-bounds handling, and inter-robot communication will be implemented. All of which will make the robots more autonomous and self-driving.

3.5.2 Localization

One fundamental requirement for this projects is to have a system for indoor localization and positioning. For real-world scenarios, Global Positioning System (GPS) is the preferred solution that fits the requirements for accuracy and latency. However, a GPS solution does not fit the requirements for this project as it has problems with indoor use and cant deliver the accuracy needed for this project. An indoor positioning system was therefore researched and tested to sense and correct robot inaccuracy.

Multiple alternatives for a GPS substitute were researched and tested during the project. Ultimately, Lane Detection provided the best results for localization and was realizable considering the time available and the scope of the project. The proposed solution for localization is not as accurate as GPS or other solutions for indoor localization on the market. The following section discusses some of the alternatives to indoor localization that was researched during the project.

Digital Compass

During the pre-project phase, experiments with a digital compass were performed. Given that the system knows the direction of the robot movement (north, south, west, east), the hypothesis was that the compass could be used to adjust the heading of the robot to match the system heading. One of the robots was equipped with a 3-Axis Digital Compass provided by Grove[3] and experiments were performed.

When testing the hypothesis, the robot was commanded to drive straight for a specified period, and sample readings from the compass was collected every 0.1

[3]https://www.seeedstudio.com/Grove-3-Axis-Digital-Compass-p-759.html

seconds. The samples were loaded into Excel and plotted in a scatter plot and analyzed. Here it was quickly realized that the compass was too affected by the noise coming from the environment. Some calibration techniques were applied to see if the noise could be reduced without any useful results. In the end, the conclusion was that a digital compass could not provide enough accuracy for this project.

Apriltag

AprilTag is a visual fiducial system, useful for a wide variety of tasks including augmented reality, robotics, and camera calibration [Ols10]. It uses software detection based on based on locating QR like codes, tags, to compute the precise 3D position, orientation, and identity of the tags relative to the camera. Multiple demonstrations[4] with tracking is available. Such a solution would benefit this project, but due to limited time, it would not be possible for this project.

Lane Detection

Lane Detection is a mechanism first used to warn drivers when they cross a lane to avoid accidents such as drowsiness. It uses computer vision to gain a high-level understanding of digital images or videos and tries to automate tasks that the human visual system can do [APH17]. This concept is applied to self-driving cars by identifying lanes and calculate the cars current heading based on the position of the lanes and make steering adjustments accordingly. Keeping the car in the middle of the lane. Interestingly, this method can be used as a tool to adjust our robots heading and increase the position accuracy of the robots. By using Lane Detection, we can assure that the robots will drive within its lanes and therefore increase the confidence that the robots physical position will match the position calculated by the software. The process of lane detection is described in detail below with figure 3.7 showing the output from each step in the algorithm.

1. The algorithm starts by processing one image captured by the Raspberry Pi Camera (figure 3.7a).
2. The captured image is then converted into grayscale to reduce noise and to better separate lane lines and road (figure 3.7b.
3. To further reduce the amount of noise in the image, Gaussian smoothing is applied to remove sharp edges and make the image smoother (figure 3.7c.
4. To detect edges (quick changes in transitions in the image) we use the Canny Edge Detection algorithm[5] provided by the OpenCV library[6]. This will detect

[4] https://www.youtube.com/watch?v=MHADQP6fV_c
[5] https://en.wikipedia.org/wiki/Canny_edge_detector
[6] https://opencv.org

structural information such as lines and help reduce the amount of data that needs to be processed (figure 3.7d).

5. We are only interested in certain parts of the image and trim the image to only focus on the region of interest. In our case the road and lane lines (figure 3.7e.

6. To convert the edges from our trimmed image we apply the Hough Transform algorithm[7] which will convert edges into points in x and y pairs. The result is an array of small lines extracted from the image.

7. We then use linear algebra to average all the lines returned from Hough Transform and create two separate lines: left lane and right lane. When averaging over all lines, we exclude lines with a small slope to remove horizontal lines.

8. We now have two lines and can use these lines to calculate the current center point in the image. This center point is then be compared to the actual image center, and the offset is used to correct the robot motor movement during simulations (figure 3.7f).

3.5.3 Simulate driving behavior

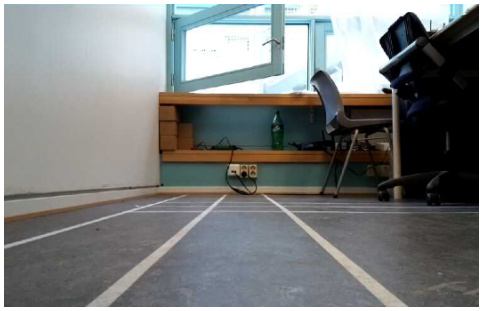
The third and final aspect is that the testbed should mimic human driving behavior. Based on the theory and assumptions we have identified three characteristics that describe driving behavior that is common for all drivers of vehicles.

1. **Route Planning**: Drivers usually drive a partially to fully planned route. The purpose of a vehicle is to transport one from point A to B by following a route. We can incorporate this in our testbed by having a module dynamically creating a route for the robots.

2. **Speed Adaption**: Acceleration and deceleration is an important aspect of regular driving. To get accurate results from simulation, the robots should accelerate after a stop and decelerate when stopping.

3. **Consider other cars**: To avoid accidents it is important that each robot know which robots are in its vicinity. This can be achieved via location beacons broadcasted in the network and sensor data from the camera.

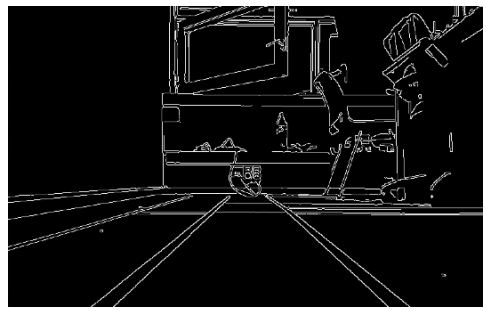
[7] https://en.wikipedia.org/wiki/Hough_transform



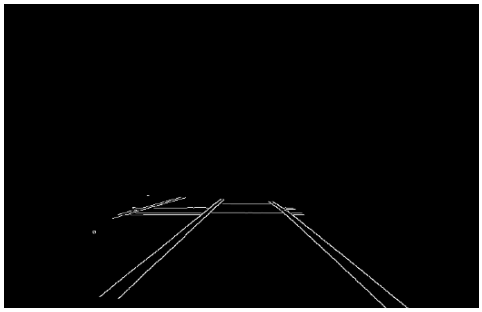
(a) Original image

(b) Gray image

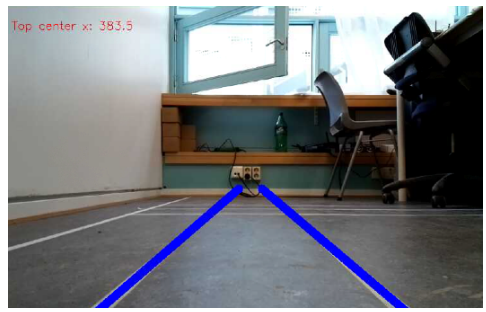
(c) Blurred image



(d) Edged image



(e) Trimmed image



(f) Final image

Figure 3.7: Output from the different steps in Lane Detection algorithm

Chapter 4
Implementation

This chapter presents the architecture and explains the modules that make the ITS testbed. The testbed is a hybrid solution for testing ITS applications in a controlled environment. Combining the benefits from computer simulations and testing on real cars at a low cost. The chapter starts with an introduction to the system followed by an overview of the system architecture, before explaining the design and implementation of each module and ending with a description of the implementation of both regular and virtual traffic lights. The resulting project is open source and located on github: https://github.com/oleaha/vtl.

4.1 Architecture

The systems architecture is depicted in figure 4.1 and consists of six separate modules at four different abstraction levels. At the top level is the planner module and is the module that is implemented to mimic the human driving style. The planner's primary goal is always to have a route for the robots to follow. At the lowest level, is the motor control module which is responsible for sending commands to the robot's motors and is the lowest abstraction level. Each of the modules, their implementation and design are described in the following sections. The car module encapsulates the whole program and is the coordinator for inter-module communication.

4.2 Planner

The planner is the module with the highest abstraction level in the system and can correspond to how a human interacts with a car. Its high-level goal is to provide the robots with a purpose during simulations. When a person uses a vehicle, the car serves the purpose of transporting the person from point A to B and the person decides which route to take. It is also a person job to interact with the car, such as turning the steering wheel, indicating a turn, acceleration, and breaking. In this system, the planner takes on the job as the human. It makes sure that the robots

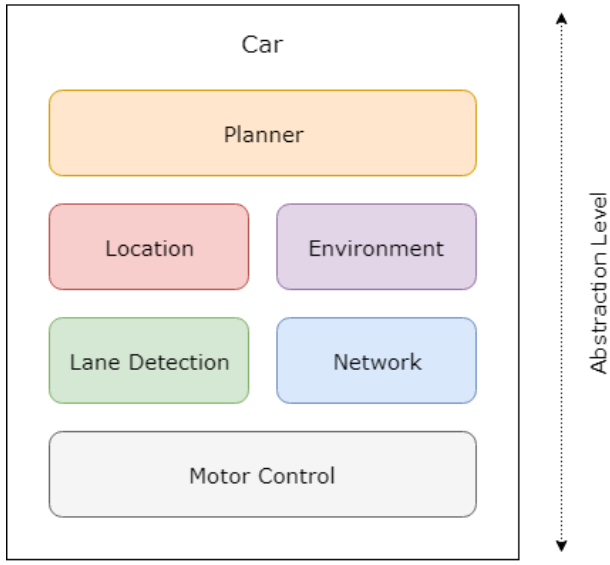


Figure 4.1: Architecture overview

always have a route to follow and give commands on when to drive straight, make a turn, accelerate and brake. The robot then takes these high-level instructions and translates them into machine-level commands. Self-driving cars, all though more complex, also uses the notion of a planner to transport people from A to B [FSYW15]. In this project, the environment and world are quite simplified as opposed to the real world. Our environment only supports straight roads with simple intersections which will reduce the complexity of the planner's implementation.

The idea of implementing a planner comes from the theory of Automated Planning and Scheduling (APS), a branch of artificial intelligence that involves the realization of strategies or action sequences [MG]. **RQ6** implies that the system needs an internal map of the environment which are also one of the assumptions stated for VTL. Since all cars have an internal map, planning can be performed offline as opposed to unknown environments where a plan often needs to be revised online. The benefit of offline planning is that a strategy can be found and created before execution. If a planner is given a description of the possible initial states of the world, a description of desired goals, and a description of the set of possible actions, the planning problem is to synthesize a plan that is guaranteed. That is when the plan is applied to any of the initial states. Creating a state containing the desired goals, our goal state.

The simplest possible planning problem, known as the Classical Planning Problem is determined by:

1. A unique known initial state
2. Durationless actions
3. Deterministic actions
4. Which can be taken only one at a time
5. A single agent

Most of these requirements are fulfilled in our system, with one exception due to robot inaccuracy explained in section 4.4. Given our classical planning problem, the world after any sequence of actions can be predicted, and the job for the planner module simplifies for this project. For the system to enforce traffic rules and guarantee full environments exploration (**RQ1** and **RQ5**)only a given number of commands are calculated by the planner each time. Figure 4.2 shows the finite state machine that describes the planner and its actions.

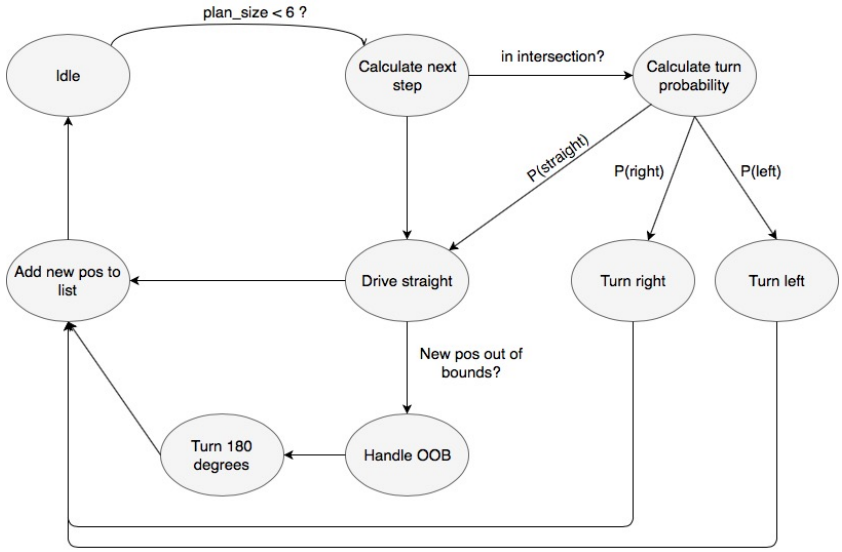


Figure 4.2: Finite state machine showing the planning process performed by the Planner.

The planner is initiated by the car module in section 4.7. Before any actions are taken, the planner creates a list of n commands which serves as the plan for the robot to perform. Each time the robot is to perform a move the first command of the planned list is removed, and its appropriate action is calculated based on the

command. As depicted in figure 4.2 there are four possible actions created by the planner: drive straight, turn left, turn right and turn 180°.

4.3 Environment

RQ1 states that the testbed should be flexible and support multiple simulation environments. To achieve this, a modular environment module is implemented. The module consists of three different classes: map, intersection, and road. Which are the entities needed for performing simulations in this application. Since all entities in the map are classes, functionality can easily be added to support different types of ITS applications and scenarios. The map class is responsible for building a two-dimensional array with coordinates and assign each coordinate its entity. Either a road, an intersection or out-of-bounds.

An example of a simulation map is shown in figure 4.3. The 1s represent a road and where a robot legally can drive. 3s represent an intersection and contains an instance object of the intersection class, enabling functionality such as regular traffic lights. 8s represent the position of the robot, and lastly, 0s represent all positions that are out-of-bounds and not valid positions for our robots. We could imagine this to be buildings, sidewalks, parks, and so on.

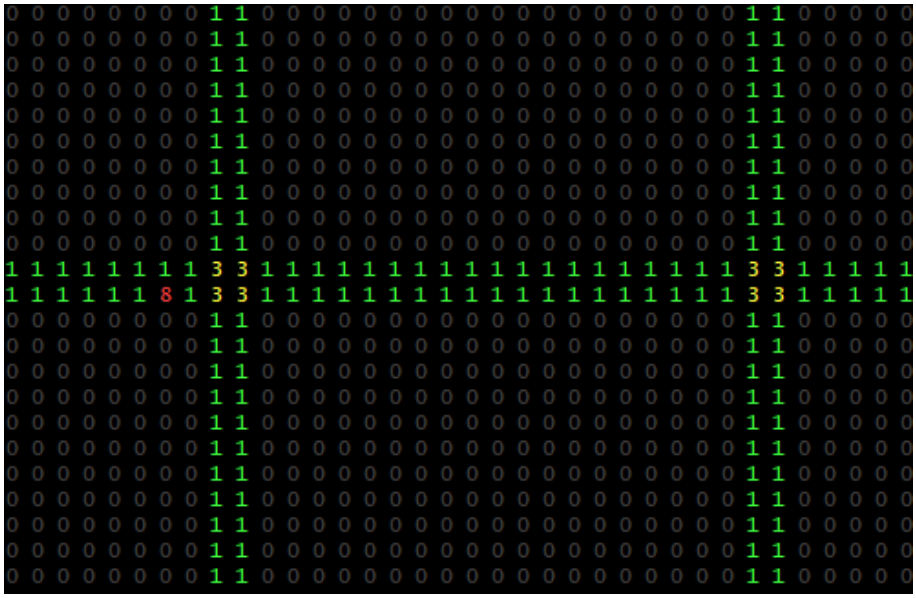


Figure 4.3: Robots internal map illustrating roads, intersections and the robots position in the system.

4.4 Location

Location module handles the location features of the system that should fulfill **RQ6**. One of the assumptions of VTL and ITS in general, is that every vehicle is equipped with a GPS with lane-level accuracy and that all vehicles share the same digital roadmap. Meaning that the system, supported by sensor data, need to guarantee the robots position within a certain range.

For the system to be accurate for this projects use case, the implementation of the location module is separated into two parts. One is responsible for the virtual location of the robot, i.e., the software position on the format (x, y). The other part is the lane detection which is responsible for correcting the robots inaccuracy in movements. Decreasing the probability of the robots virtual location to deviate from its physical position in the environment. Each module is explained below.

4.4.1 Virtual Location

As mentioned earlier, the virtual location module is responsible for all traffic logic and the robots virtual position. For this project, the following traffic logic has been implemented:

- **Enforcing right-hand side driving**: Make sure that the robot follows the regular traffic pattern of driving on the right-hand side. This also includes how the robot should handle an intersection when performing either a left or right turn
- **Out-of-bounds handling**: When the robot tries to drive outside of its internal map, a procedure to turn the car around to the correct lane is enforced.
- **Distance to/from intersection**: Functionality to check how close a robot is to an intersection

This module is mainly used by the Planner to create commands for the robot to execute. It is also sometimes used by the Car module to get information about its current position. This is due to the practicality of the testbed supporting multiple types of ITS applications.

4.4.2 Lane Detection

The idea of Lane Detection is explained in detail in section 3.5.2. Its main goal is to make sure that the virtual position does not deviate from the robots actual position. Lane Detection uses OpenCV, an open source computer vision library.

Running the following command on the Raspberry Pi devices installs OpenCV and its dependencies:

```
$ sudo apt-get install python-opencv
```

Due to the robots previously mentioned inaccuracy, lane detection is used to reduce this inaccuracy, decreasing the probability of the virtual position not being the robots physical position on the map. To guarantee performance, Lane Detection is executed as its own process by using the Python Multiprocessing library[1]. As soon as the system starts, lane detection starts to collect the current heading used for adjustments. All collected points are added to a multiprocessing queue to avoid deadlocks and synchronization issues when communicating among processes. The samples provided by the algorithm is then used in the Motor Control module (section 4.6) to make the final adjustment.

4.5 Network

A central aspect of the testbed is the V2V capabilities stated in **RQ3**. In this project, the network topology and the network module together serve as the replacement for VANET and DSRC. The following subsections describe the network topology and the onboard network module that provides communication.

4.5.1 Network Topology

Raspberry Pi 3 comes with an onboard wireless interface used for sending and receiving packets. IEEE 802.11n provides ad-hoc functionality which serves as our VANET implementation. To enable ad-hoc functionality some configuration on the Raspberry Pis are needed. The configuration is done by editing the network interface file located at */etc/network/interfaces* to include the settings shown in figure 4.4. In order to connect to the robots remotely, we also need to install a DHCP server to assign IP-addresses. On the Raspberry Pi, we install the ISC open source DHCP software system[2] and edit the *dhccp.conf* file located at */etc/dhcp* to include the configuration shown in figure 4.5.

After configuring all devices in our testbed with the same settings, assigning each Raspberry Pi with a unique, static IP-address, robots can communicate over V2V in an ad-hoc manner. Notably, this enables the devices to communicate with every device in its vicinity. Figure 4.6 shows the topology of the ad-hoc network in this project.

[1] https://docs.python.org/2/library/multiprocessing.html
[2] https://www.isc.org/downloads/dhcp/

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp

auto wlan0
iface wlan0 inet static
    address 192.168.1.1 # This address has to be unique
    netmask 255.255.255.0
    wireless-channel 1
    wireless-essid its_testbed
    wireless-mode ad-hoc
```

Figure 4.4: Network configuration.

```
ddns-update-style interim;
default-lease-time 600;
max-lease-time 7200;
authorative;
log-facility local7;

subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.5 192.168.1.100;
}
```

Figure 4.5: DHCP configuration.

4.5.2 Network Module

The Network Module is the module responsible for creating, sending and receiving packets to and from other robots. In our scenario, the network module acts as the DSRC device or Onboard Unit (OU) of the robot. The module is separated into two logical parts: one for sending and one for receiving messages, providing different functionality to the system.

The concept of VANET and VTL relies on the use of broadcast messages in the network to exchange beacons. The system also needs to support different types of messages. In this projects, there are two available message types: one for periodically sending location beacons and one that is application specific. Since this testbed is to be a general purpose testbed for ITS applications, implementation of application specific messages needs to be developed to fit the ITS application being tested. An example implementation has been done for a centralized regular traffic light and a decentralized implementation of the VTL algorithm and is described in section 4.8.

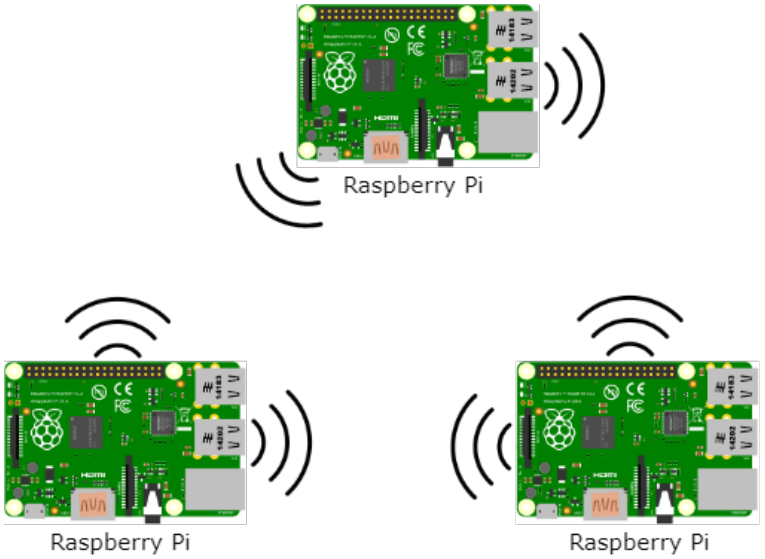


Figure 4.6: Ad hoc network with Raspberry Pi 3 devices

Python sockets[3] are used to achieve the described functionality. We bind a socket to an IP-address and port for sending and receiving packets over the network. For sending and receiving messages the User Datagram Protocol (UDP) is used. It was chosen primarily for its quick implementation and requires less overhead compared to TCP and is the protocol used in VANET. To make a socket able to broadcast messages, the socket is bound to the network broadcast address. In our network setup, the broadcast address is 192.168.1.255. Upon receiving messages from the network, a server socket listening to messages sent to its IP-address and port is established.

Figure 4.7 shows a sequence diagram explaining how a broadcast message is sent through the ad-hoc network and received at all connected nodes. Raspberry Pi A is the sender of the message to the broadcast address. A message sent to this address will be delivered to all connected nodes in the range of the source. For the receivers - Raspberry Pi B and C - the message seems to come directly from Raspberry Pi A. Figure 4.8 shows the same functionality only for application specific messages, adding the option of replying to a broadcast message used as a request.

[3]https://docs.python.org/2/library/socket.html

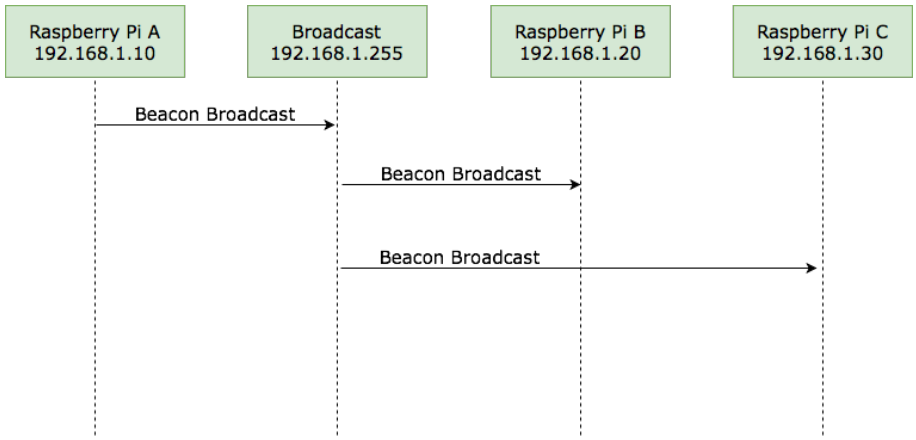


Figure 4.7: Sequence diagram showing the flow of a broadcast message

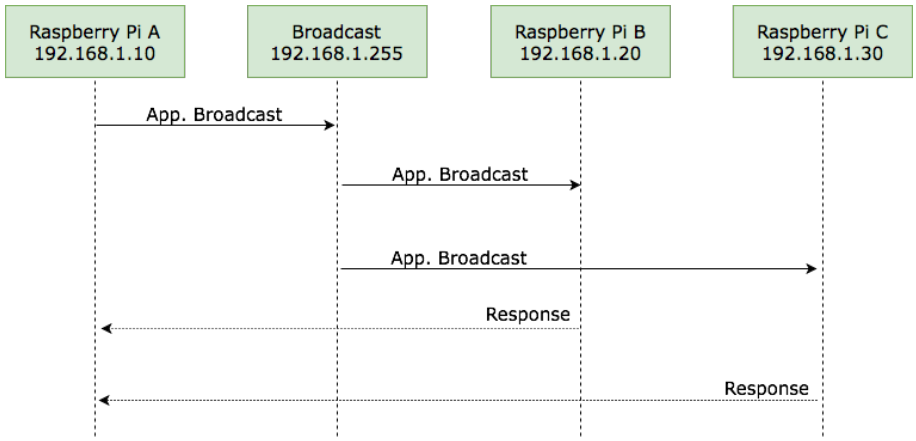


Figure 4.8: Sequence diagram showing the flow of a application specific message with reply

4.6 Motor Control

The Motor Control module is the lowest level in the project stack. It translates higher level commands into robot-readable instructions. To send these commands to the hardware, an API provided by the PiBorg Organization is used[4]. Through this API one sets the power and direction for each of the two motor sets. One of the main drawbacks of the DiddyBorg robots is that there is no confirmation of work done, such as a rotary encoder that can be used as a guarantee that the robot moved according to its command. Meaning that every movement is based on sending a constant voltage to the motors for a timed period. Current solution and best practice provided by PiBorg is to measure the time the robot uses to drive 1 meter and then use this as a guideline for all movements. If the robot is to drive 20cm the time needed is $0.2m * timeForwardOneMeter$. The same applies when a robot performs a turn. We measure the time taken to perform a 360°spin and use this measure for all turns. Eventually, this method is the cause of the system's inaccuracy concerning movements. Many factors affect the movement when movements are time-based rather than telemetry based. Factors such as friction and battery capacity are the primary cause of inaccuracy in movements. To correct these movements the Motor Control module uses the values provided by the Lane Detection module. If the current center value provided by Lane Detection deviates from the actual image value, we increase or decrease the power sent to one of the motor sets based on the value.

4.7 Car

Finally, the car module is the main module of the system that ties all modules together. It is also the module that performs simulations and where code can be added to test different types of ITS application. An exciting and challenging design goal for this module is to keep it as simple as possible and hide the functionality of the testbed as best as possible. The idea behind this is to make it easier to learn to use the system and decrease the implementation time for testing new applications. When starting and performing simulations, the car module is the main entry point for the current robot. The command to start a simulation is:

```
$ python simulation/car.py <ip-addr> <pos> <dir>
```

This command includes all initial information the system needs in order to perform simulations. When this command is executed, an initialization process is started. This process will take care of setting up all modules and threads with the correct information. After the initialization is performed the program enters a while loop that runs until a user terminates the program. For each iteration, it will check the

[4]https://www.piborg.org/blog/build/thunderborg-build/thunderborg-examples

planned queue and take the next command. Based on the command, the appropriate action is taken and some sanity checks to avoid collisions with other cars is also performed. It is important to mention that the plan provided by the planner is not the absolute truth. The car itself needs to make sure that it can perform the command provided by the planner. It needs to avoid collisions, obey traffic rules enforced by traffic lights or virtual traffic lights.

4.8 Applications

To evaluate the performance of the testbed, two example applications were developed during the project. The first application is a regular, centralized traffic light and the second is a decentralized implementation of the VTL application. In chapter 5 these applications are used to perform simulations, evaluate the testbed and the effectiveness of VTL.

4.8.1 Regular Traffic Light

To implement a regular traffic light, a separate Raspberry Pi was used as the controlling unit of an intersection. The purpose of the Raspberry Pi is to communicate the current state of the intersection to the robots entering an intersection. Since we do not have any physical indicator of the intersection state, the system transmits the state over the network as broadcast messages and the robots use these messages to determine if it is presented with a red or green light.

All intersections in the environment run their own instance of a regular traffic light. When a traffic light is started, it follows a pre-determined timed cycle and runs until the system is stopped. The intersection implemented in this project alternates between two states:

1. Green light for robots approaching from north and south
2. Green light for robots approaching from west and east

The switch between states is triggered when the green light timer T_{green} expires. The traffic light then sends out a yellow traffic light for the current state, allowing robots in the intersection to pass. After the yellow timer T_{yellow} expires, T_{green} is restarted and the state updated. If there is no switch between state, the traffic light sends out a green traffic light for the current state. Every time a robot enters an intersection is checks a traffic light state array and is presented with either a green or red light. The receiver thread updates the traffic light state array, and a message handler identifies the type of message and updates the traffic light state array. The

different timers that define a traffic light cycle can be changed to match different scenarios in the provided settings file in the project.

4.8.2 Virtual Traffic Light

The VTL idea and basic operation were introduced in section 2.4 and provide the basic understanding needed to understand the implementation used in this project. Our implementation is highly based on the distributed VTL algorithm proposed by A. Bazzi et al. in [BZMP14]. The scope of this project is to create a testbed for ITS and due to time constraints and complexity, some of the features proposed in [BZMP14] is omitted to ease implementation time. The algorithm is hereafter described through a procedure which is followed by a generic (leader or follower) robot, denoted R_a. VTL_a indicates the VTL of R_a. Table 4.1 describes the different messages sent over the network during the VTL procedure. The VTL algorithm starts when R_a enters the VTL area which is a pre-defined distance from an intersection and considers only robots that are within the VTL area.

1. VTL_a is set to Yellow.
2. R_a evaluates its location table and find all robots that are within VTL area for the current intersection. The set of robots are denoted H. If $|H| = 0$, then there are no conflicts and VTL_a is set to Green
3. If $|H| > 0$, sort H based on distance to the intersection. If no robots have a distance of 0 to the intersection, the algorithm is restarted. If at least one robot has a distance of 0 to the intersection, the procedure continues.
4. Determine if R_a is a leader or follower on the current road. If R_a is a follower it is presented with a red light and the algorithm restarts.
5. If R_a is a leader and R_a is closest to the intersection based on the sorted set H, then R_a sends a GRR request message to all robots in H. R_a then waits for an ACK message for all robots in H. When all ACKs are received, VTL_a is set to Green. If R_a is a leader but not closest to the intersection VTL_a is set to Red and the procedure restarts.

At any given time a robot can receive a GRR request which it must answer with an ACK. This deviates from the implementation in [BZMP14] where a robot can answer with a NACK if it has already answered to a GRR request in the current VTL operation. Due to the scale and design of the testbed, this scenario is not likely to happen given the simulation setup described in chapter 5 and was therefore not included in the VTL implementation.

Message	Message Code	Scope	Destination	Parameters
Green Request	GRR	Ask for a confirmation when the robot calculates it has the priorty	All robots in H	Origin, Checksum, Intersection ID
Acknowledgement	ACK	Confirm a GRR	The robot that sent a GRR	Receiver, Checksum, Origin, Intersection Id

Table 4.1: V2V messages sent during VTL procedure.

Chapter 5
Simulation

In this chapter, we will perform simulations on the testbed to see how it performs with respect to the requirements and its usability. First, the simulation setup is described together with the metrics that will be collected during the simulations. We then define three different simulation scenarios that will be performed on the two implemented ITS applications as described in section 4.8: regular traffic light and virtual traffic light. Finally, we compare results from the simulations to evaluate the testbed and the efficiency of VTL.

5.1 Simulation setup

Each application is tested on three different scenarios. Each scenario consists of two or more DiddyBorg robots as presented in section 3.3.1 with the testbed implementation installed on each robot. The difference between the simulations are in the environment and the number of robots used. All robots use the same configuration settings (described in appendix B) that the scenario or application needs in order to perform its task. For all simulations, one movement for the robots is defined as being a movement of 25 centimeters. During simulations, metrics defined in table 5.1 are being logged to a log file for post-simulation analysis. All scenarios use the same stopping criteria which are when all cars have passed an intersection 60 times. The following subsections define the different simulation scenarios. Figure 5.1 show the testbed in action. A short video showing the early testing stage of the testbed is located here[1].

5.1.1 Simulation Scenarios

Three different scenarios are designed for this thesis to observe the effect number of robots and intersections have on regular and virtual traffic lights.

[1] https://youtu.be/Ojllqlysp3g

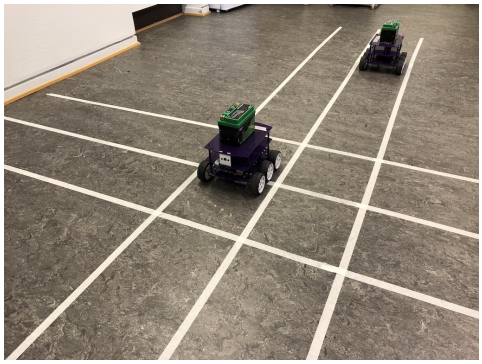
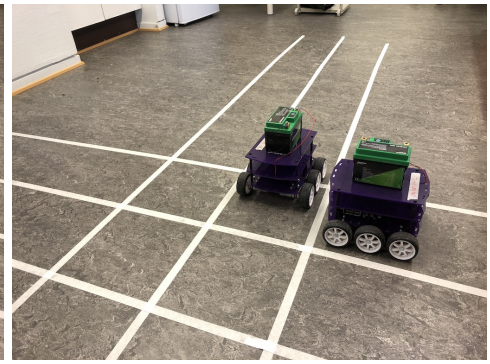



Figure 5.1: Images from testbed in action

Metric	**Definition**	**Unit**
Number of crossings	The total number of times a robot passes through an intersection independent on the application used in that intersection.	Integer
Number of steps	The total number of steps a robot drives during a simulation. All robots are configured to drive 25 cm for each step, giving us the total number of meters driven during a simulation.	Meters
Simulation time	The total time from start to end in seconds.	Seconds
Average speed	The average speed of the robot during the whole simulation derived from dividing the number of steps with the total simulation tim.e	m/s
Wait time	The amount of time spent waiting for a green light in an intersection.	Seconds
Average wait time	Total wait time divided by the number of crossings.	Percentage
Wait percentage	The percentage of time spent waiting during the whole simulation. The total wait time divided by total simulation time.	Percentage
Queuing time	Amount of time spent queuing in intersection. There is another car in front.	Seconds

Table 5.1: List of metrics collected during simulations.

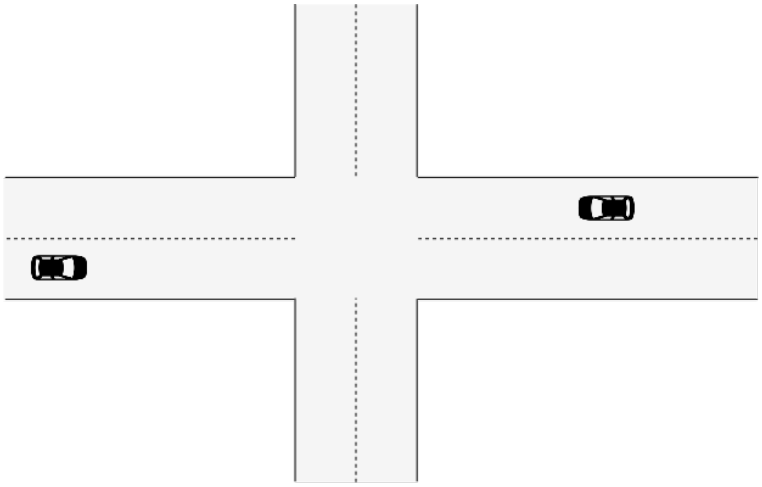


Figure 5.2: Simulation scenario 1

Scenario 1

The first scenario (figure 5.2) uses two DiddyBorg robots and one single intersection governed by either a regular or a virtual traffic light. The map creates a 12×8 grid which covers an area of $6m^2$ and a total road area of $2,25m^2$. Both robots combined use an area of $1/8m^2$ yielding a car-to-road ratio of 1/18.

Scenario 2

The second scenario (figure 5.3) uses the same layout as in scenario 1 with an addition of two DiddyBorg robots giving a car-to-road ratio of 1/9. The purpose of this scenario is to see how both regular and virtual traffic lights behave when the density of vehicles increases.

Scenario 3

The third scenario (figure 5.4) introduces a new environment with two intersections and four DiddyBorg robots. The map is updated to a 12×10 grid with a total area of $7,5m^2$ and a total road area of $3,5m^2$ giving a car-to-toad ratio of 1/14. The purpose of this scenario is to see how the testbed handles multiple intersections and how both regular and virtual traffic lights will perform.

5.2 Regular Traffic Light Simulation

Simulations performed with regular traffic lights use the implementation described in section 4.8.1. The regular traffic light solution follows a pre-determined cycle of

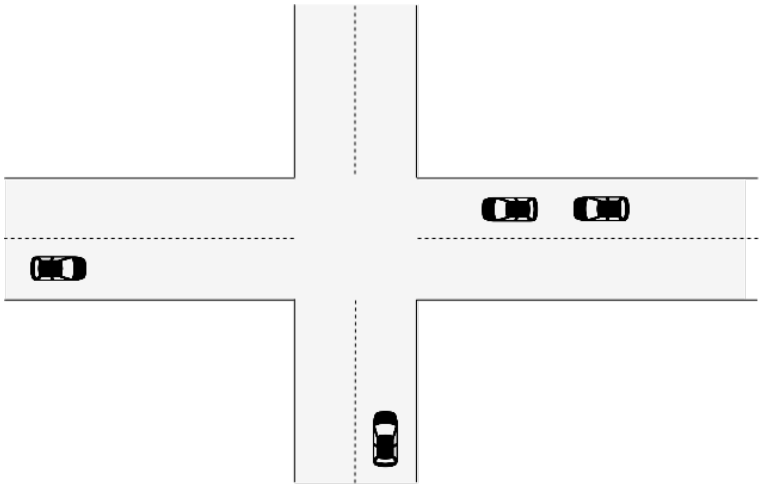


Figure 5.3: Simulation scenario 2

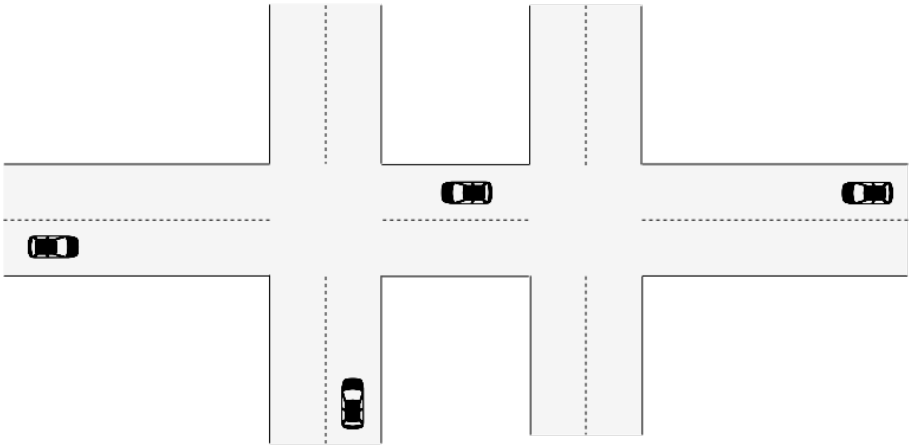


Figure 5.4: Simulation scenario 3

40 seconds: 15 seconds of green light, 5 seconds of yellow light and 20 seconds of red light. The robots are not allowed to enter the intersection during the yellow light phase. This is to ensure that all robots have enough time to cross the intersection before the intersection enters the red light phase.

In order to communicate intersection state, an independent Raspberry Pi is used at each intersection. This Raspberry Pi takes the role of a physical traffic light serving as a roadside unit. Intersection state is broadcasted on the network for robots to know the state of an intersection. Results collected from the three different scenarios are summarized in table 5.2. The full results can be seen in appendix A.

Regular Traffic Light			
	Scenario 1	Scenario 2	Scenario 3
Average speed (m/s):	0,0666	0,0659	0,0539
Average wait time in intersection (s):	8,65	10,48	14,39

Table 5.2: Results from regular traffic light simulations

5.3 Virtual Traffic Light Simulation

Simulations performed with VTL application uses the implementation described in section 4.8.2. Robots communicate when approaching an intersection to check for and resolve possible conflicts. The VTL application is tested on the same scenarios as regular traffic light and results collected from the scenarios are summarized in table 5.3. The full results can be seen in appendix A.

Virtual Traffic Light			
	Scenario 1	Scenario 2	Scenario 3
Average speed (m/s):	0,1097	0,1038	0,1063
Average wait time in intersection (s):	0,38	0,57	0,63

Table 5.3: Results from virtual traffic light simulations

5.4 Results

To compare the results from regular traffic light and virtual traffic light there are two metrics of interest: the average speed during simulation and the average wait time in an intersection. The VTL protocol is a traffic optimization algorithm that tries to reduced time stopped in traffic in order to increase traffic flow and reduce congestion. Figure 5.5 show the average wait time that a robot waits for a green light at an intersection. The difference between the regular timed traffic light and

the virtual traffic light is quite significant with a reduction in average wait time around 90%. Comparing this results with simulations performed in other studies ([FFCa+10], [CFS13]), our result seems to be a little optimistic for an urban scenario. This is mainly due to the density of vehicles at the intersection. If the density is low - as in the simulations performed - the reduction in wait time is due to the elimination of unnecessary red lights enabled by the VTL protocol. The reduction on wait time correlates to the average speed of a robot during simulations. Figure 5.6 shows an average speed increase of around 71%. The effectiveness of VTL in low-density scenarios is promising, for higher-density scenarios it is expected that the average wait time will increase but still be lower than traditional traffic lights as stated by M. Ferreira et al. in [FFCa+10]. An unexpected feature provided by the system is the option to add virtual robots when performing simulations providing an option to test scenarios with a higher density of robots. This could be done by deploying the code to a set of Raspberry Pis and mocking the module that sends commands to the motors but otherwise functions as a regular robot. However, due to limited time, this was not done.

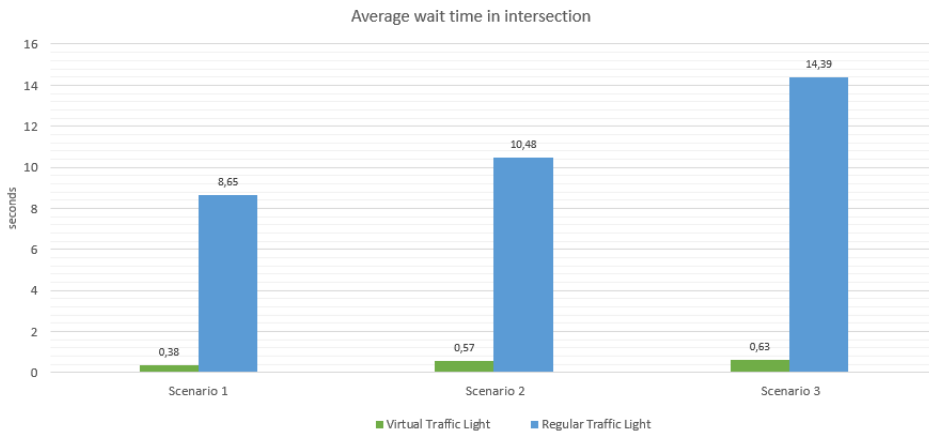


Figure 5.5: Graph displaying the average wait time for a green light in an intersection for all three scenarios.

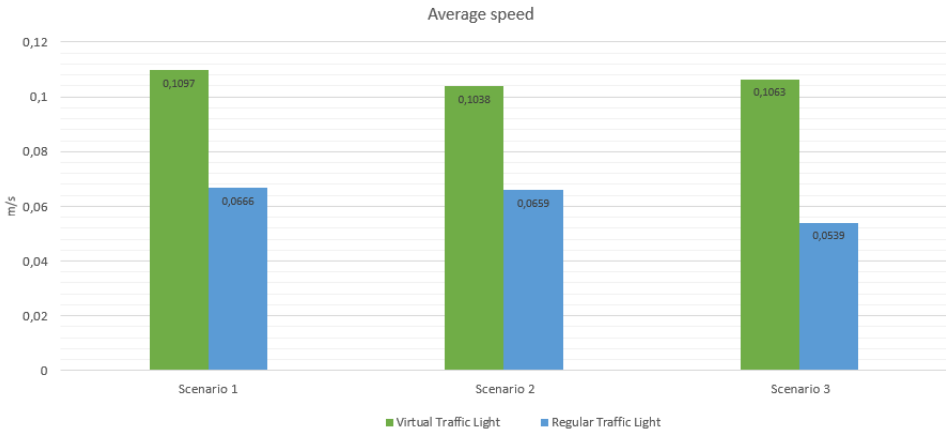


Figure 5.6: Graph displaying the average speed during the simulations for all three scenarios.

Chapter 6
Evaluation, Conclusion and Future Work

6.1 Evaluation

The work performed in this thesis is motivated by the need for Intelligent Transportation Systems to better exploit our current road infrastructure with new traffic optimization solutions. This is made possible by utilizing technology advancements within information and communication technologies and self-driving cars. The primary objective of this thesis is to create a low-cost ITS testbed and give research and developers the possibility of performing tests closer to a real-world scenario as compared to computer simulations. To create such a solution seven requirements where identified and serves as the main criterias for the testbed.

Based on the simulations performed, the implementation itself was evaluated on the expected behavior defined by the requirements. The evaluation of these requirements is based on observations made during the simulations as it turned out to be difficult to find quantitative metrics to evaluate the implementation. Thus the method used for evaluation is a qualitative approach based on the requirements. Areas of interest include robot movement accuracy, enforcement of traffic rules and communication between robots. Overall, based on the observations, the system behaves as expected within the given requirements. There is, however, some exceptions that we discuss further.

RQ5 states that: "Robots should enforce regular traffic rules and behavior such as driving on the right-hand side, be aware of other robots and adapt speed according to the situation." During simulations the speed adaption mechanism provided problems and ended up being removed from the testbed implementation. The robots used in this testbed uses motors that need a constant stream of power when moving and a movement is time-based. Dynamically changing the power sent to the motors turned out to be challenging and time-consuming without yielding expected results. The problems of speed adaption propagated through the system and ended up affecting the localization part where the system position did not match the robots physical

position.

As mentioned, indoor localization turned out to be a challenge for this testbed. **RQ6** states that the robots should have lane-level accuracy and an indoor supplement for GPS is needed. In the implementation, lane-level accuracy is supported by two parts: virtual position and lane detection. The initial hypothesis was that lane detection could be used as a mechanism to correct the robot's movement and thereby reducing the inaccuracy of position miss-match between the system and the robots physical position. As observed throughout the simulations, the lane detection correction mechanism implemented in the Motor Control module not fully optimized to fulfill the requirement. Resulting in inaccurate movements and manual adjustments of the robots were needed during the simulations - as seen in the video[1]. When the robot drives in a straight line, the lane detection mechanism manages to adjust the robots heading correctly. Problems occurred when the robots performed a turn, either in an intersection or when reaching the end of the map when the out-of-bounds procedure is initiated. When a turn was inaccurate, possibly due to reduced battery power, the lane detection mechanism would not function properly resulting in position miss match. However, there is a belief that with further optimization of the correction mechanism, the proposed solution will confirm the hypothesis and achieve lane-level accuracy.

Concerning flexibility and scalability of the testbed some constraints for **RQ1** and **RQ2** apply. The implemented system support changes in the environment but restricts the type of changes to simple intersections and straight roads. One change in position on the map correlates to a movement of 25cm that possibly can put some restrictions on the types of ITS applications tested on the system. During simulations, a total of four DiddyBorg robots were used. Based on observations, there should not be any problems to scale up the number of robots since each robot are independent and communication is done over the ad-hoc wireless network.

The only non-functional requirement specified is that the system should have a low cost, be easy to set up and to perform simulations. The total cost of the testbed depends on the number of robots needed. The total cost of one robot is close to $350. Comparing the cost to other relevant testbed implementation presented in section 2.6 the cost is similar to other implementations. Keeping in mind that other discussed implementations often are more specific compared to the system designed in this thesis. In order to perform simulations, only the configuration file needs change which makes simulations easy and quick to perform. There is, however, need for physical lanes for lane detection to work. This is achieved by using tape to draw the map layout on the ground as seen in figure 5.1.

[1] https://youtu.be/Ojllqlysp3g

Throughout the work of this thesis multiple challenges arose. As mentioned above, localization and robot movement inaccuracy turned out to be one of the toughest challenges in this thesis. Few similar solutions of a hybrid testbed exist, which resulted in a lot of researching and experimenting to find solutions that often are a trade-off between complexity and accuracy, as with lane detection. At times it was frustrating to find and apply solutions in other fields and trying to implement them in the system. On a positive note, since the applied methodology is a closed learning loop (section 3.2), many ideas were abandoned at an early stage in an iteration which saved time. Overall, the system described in this thesis can deliver valuable results and insights for researchers and developers. The combination of low-cost robots with sensor data and computer simulations have the potential to save time and cost compared to field tests.

6.2 Conclusion

In this thesis, an implementation of a low-cost hybrid testbed for ITS applications are designed and implemented. What separates the proposed system from other related work is that the testbed is designed to be more a more general testbed compared to related work. By making the testbed general, the types of applications that can be tested is limited by the requirements rather than limited by a specific use-case. Throughout the thesis a closed learning loop has been applied to research, test, validate and gain knowledge within specific areas - allowing different solutions to be tested and implementing the solution that fit the objectives. The system is implemented on DiddyBorg robots by using Raspberry Pi as the controlling unit. V2V communication is enabled through the use of ad-hoc networks where robots exchange location beacons and application specific messages, enabling collaboration between robots. The proposed indoor localization strategy uses lane detection, a technology applied to self-driving cars, to correct inaccurate movements from the robots. This solution shows promising results without being fully optimized.

Simulations performed on the testbed and analyzing the results show that the system behaves as expected, although the effectiveness of the VTL protocol cant be thoroughly evaluated due to low vehicle density. However, as stated by M. Ferreira et al. when applying the VTL protocol to low-density areas, the reduction in average wait time in intersections is due to the mitigation of red lights at an intersection. A statement that is further confirmed by the simulations performed in this thesis. Based on the results and evaluation, the testbed will give researchers and developers the option of conducting real-world scenario tests at an earlier stage in a project's life-cycle.

6.3 Future Work

Future work includes improving the indoor localization procedure of the system and achieve lane-level accuracy without manual interference. This can be achieved by further optimization of the current movement correction mechanism or replacing the localization module with a more suitable solution. Another feature that can be added to the system is virtual robots to increase vehicular density without investing in DiddyBorg robots. Virtual robots will behave as regular DiddyBorg robots by the system but are not present in the physical part of the testbed. This will enable the option of performing tests with different vehicle density to see how an application behaves as density increases. Future work should also include improving the initial requirements based on the evaluation in section 6.1. The robots used in the testbed can also be extended to include more sensors such as an ultrasonic sensor for distance measurement.

References

[APH17] M. V. G. Aziz, A. S. Prihatmanto, and H. Hindersah. Implementation of lane detection algorithm for self-driving car on toll road cipularang using Python language. In *2017 4th International Conference on Electric Vehicular Technology (ICEVT)*, pages 144–148, October 2017.

[BKV+09] S. Biddlestone, A. Kurt, M. Vernier, K. Redmill, and U. Ozguner. An indoor intelligent transportation testbed for urban traffic scenarios. In *2009 12th International IEEE Conference on Intelligent Transportation Systems*, pages 1–6, Oct 2009.

[Bra17] Anders Brastad. Devlopment of a virtual traffic light testbed. Master's thesis, Norwegian University of Sience and Technology, February 2017.

[BZMP14] A. Bazzi, A. Zanella, B. M. Masini, and G. Pasolini. A distributed algorithm for virtual traffic lights with ieee 802.11p. In *2014 European Conference on Networks and Communications (EuCNC)*, pages 1–5, June 2014.

[CaDFB08] Hugo Conceição, Luís Damas, Michel Ferreira, and João Barros. Large-scale simulation of v2v environments. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 28–33, New York, NY, USA, 2008. ACM.

[CFS13] H. Conceição, M. Ferreira, and P. Steenkiste. Virtual traffic lights in partial deployment scenarios. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 988–993, June 2013.

[DD10] G. Dimitrakopoulos and P. Demestichas. Intelligent transportation systems. *IEEE Vehicular Technology Magazine*, 5(1):77–84, March 2010.

[EZL14] E. C. Eze, S. Zhang, and E. Liu. Vehicular ad hoc networks (vanets): Current state, challenges, potentials and way forward. In *2014 20th International Conference on Automation and Computing*, pages 176–181, Sept 2014.

[Fd12] 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.

[FFCa+10] Michel Ferreira, Ricardo Fernandes, Hugo Conceição, Wantanee Viriyasitavat, and Ozan K. Tonguz. Self-organized traffic control. In *Proceedings of the Seventh ACM International Workshop on VehiculAr InterNETworking*, VANET '10, pages 85–90, New York, NY, USA, 2010. ACM.

[FSYW15] M. Fu, W. Song, Y. Yi, and M. Wang. Path planning and decision making for autonomous vehicle in urban environment. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 686–692, Sept 2015.

[Hø15] Alena Høye. Safety effects of signalized intersections. Technical Report 1396, Institute of Transport Economics, 2015.

[KEBB12] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, December 2012.

[LLD+12] H. M. La, R. S. Lim, J. Du, S. Zhang, G. Yan, and W. Sheng. Development of a small-scale research platform for intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1753–1762, Dec 2012.

[MG] Paulo Traverso Malik Ghallab, Dana Nau. Automated planning and acting | Artificial intelligence and natural language processing.

[MT06] J. C. McCall and M. M. Trivedi. Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation. *IEEE Transactions on Intelligent Transportation Systems*, 7(1):20–37, March 2006.

[NAT+12] Till Neudecker, Natalya An, Ozan K. Tonguz, Tristan Gaugel, and Jens Mittag. Feasibility of virtual traffic lights in non-line-of-sight environments. In *Proceedings of the Ninth ACM International Workshop on Vehicular Inter-networking, Systems, and Applications*, VANET '12, pages 103–106, New York, NY, USA, 2012. ACM.

[Ols10] Edwin Olson. Apriltag: A robust and flexible multi-purpose fiducial system. Technical report, University of Michigan APRIL Laboratory, May 2010.

[otEU10] Council of the European Union. Directive 2010/40/eu, 2010. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2010:207:0001:0013:EN:PDF.

[TTB+15] D. Tran, E. Tadesse, P. Batapati, W. Sheng, and L. Liu. A cloud based testbed for research and education in intelligent transportation system. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 452–457, Dec 2015.

[VE03] A. Vahidi and A. Eskandarian. Research advances in intelligent collision avoidance and adaptive cruise control. *IEEE Transactions on Intelligent Transportation Systems*, 4(3):143–153, Sept 2003.

[ZYY+17] W. Zhou, T. Ying, L. Yang, Y. Yang, J. Yuan, and M. Du. Design of an intelligent driving system simulation platform and its application. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7, Nov 2017.

Appendix A
Simulation Results

Regular Traffic Light						
Scenario 1						
	Car A	Car B	Car C	Car D	Average	Std. Dev
Average speed:	0,0656	0,0677			0,0666	0,001
Average wait time:	11,2	9,5			10,35	0,85
Scenario 2						
	Car A	Car B	Car C	Car D	Average	Std. dev
Average speed:	0,058	0,0669	0,0651	0,0737	0,0659	0,0055
Average wait time:	14,5	9,6	10,25	7,6	10,48	2,51
Scenario 3						
	Car A	Car B	Car C	Car D	Average	Std. dev
Average speed:	0,0648	0,05	0,0523	0,0486	0,053	0,006
Average wait time:	8,6	16,3	15,2	17,48	14,39	3,44

Table A.1: Results collected from all robots from the different simulation scenarios with regular traffic light.

Virtual Traffic Light						
Scenario 1						
	Car A	Car B	Car C	Car D	Average	Std. Dev
Average speed:	0,1106	0,1088			0,1097	0,0009
Average wait time:	0,38	0,31			0,3495	0,001
Scenario 2						
	Car A	Car B	Car C	Car D	Average	Std. dev
Average speed:	0,1056	0,0981	0,1083	0,1034	0,1038	0,004
Average wait time:	0,43	1,25	0,03	0,566	0,570	0,49
Scenario 3						
	Car A	Car B	Car C	Car D	Average	Std. dev
Average speed:	0,1072	0,1089	0,1023	0,1065	0,1063	0,002
Average wait time:	0,51	0,35	1,1	0,58	0,63	0,28

Table A.2: Results collected from all robots from the different simulation scenarios with virtual traffic light.

Appendix B
Configuration settings

```
"""
Define the map layout constants to build the map.
Number of cells in X and Y direction. Which cells are roads and
where intersections are to be placed
"""
MAP_SIZE_X = 12
MAP_SIZE_Y = 8

ROADS_X = [3, 4]
ROADS_Y = [5, 6]

INTERSECTIONS = [
    ((3, 5), (3, 6), (4, 5), (4, 6))
]

"""
Define the probabilities of performing a turn in an intersection.
P(left) = 0.3
P(right) = 0.3
P(straight) = 1 - P(left) - P(right)
"""
PROBABILITIES = {'left': 0.30, 'right': 0.60}

"""
Constants used for ThunderBorg to calibrate the robots movement
"""
QUARTER_TURN_DEGREES = 90
HALF_TURN_DEGREES = 180
DRIVE_STEP = 0.25  # 20 centimeters per step
```

```
"""
Lane detection settings describing frame rate,
resulution and the center point in the image
"""
CAMERA_FRAME_RATE = 3
CAMERA_RESOLUTION = (752, 480)
CAMERA_VFLIP = False
CAMERA_HFLIP = False
CAMERA_WARMUP_TIME = 0.1
ACTUAL_CENTER = 383
LANE_DEBUG = False

"""
Definition of a regular traffic light cycle in seconds
"""
TRAFFIC_LIGHT_INTERVAL = 12
TRAFFIC_LIGHT_YELLOW_DURATION = 3
CAR_IS_PRIMARY = True
TRAFFIC_LIGHT_BROADCAST_PORT = 5555
```

Appendix C

Available ThunderBorg commands

- **RawWrite()** Sends a raw command on the I2C bus to the ThunderBorg
- **RawRead()** Reads back from the ThunderBorg after sending a GET command
- **InitBusOnly()** Prepare the I2C driver for talking to a ThunderBorg on the specified bus and I2C address.
- **Init()** Prepare the I2C driver for talking to the ThunderBorg
- **SetMotor2()** Sets the drive level for motor 2, from +1 to -1
- **GetMotor2()** Gets the drive level for motor 2, from +1 to -1
- **SetMotor1()** Sets the drive level for motor 1, from +1 to -1
- **GetMotor1()** Gets the drive level for motor 2, from +1 to -1
- **SetMotors()** Sets all motors to stopped, useful when ending a program
- **SetCommsFailsafe()** Sets the system to enable of disable the communication failsafe. The failsafe will turn the motors off unless it is commanded at least once every 1/4 of a second.
- **GetCommsFailsafe()** Read the current system state of the communications failsafe.
- **GetDriveFault1()** Reads the motor drive fault state for motor #1. Faults may indicate power problems, such as under-voltage and may be cleared by setting a lower drive power.
- **GetDriveFault2()** Reads the motor drive fault state for motor #2. Faults may indicate power problems, such as under-voltage and may be cleared by setting a lower drive power.