



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA  
(ICAI)

GRADO EN INGENIERÍA ELECTROMECAÁNICA  
Especialidad Electrónica

# **SOFTWARE INTEGRATION AND SIMULATION OF AUTONOMOUS VEHICLES**

Author: Marta Bravo Lázaro

Director: Knut Åkesson

Madrid

July 2018



# **AUTHORIZATION FOR DIGITALIZATION, STORAGE AND DISSEMINATION IN THE NETWORK OF END-OF-DEGREE PROJECTS, MASTER PROJECTS, DISSERTATIONS OR BACHILLERATO REPORTS**

## ***1. Declaration of authorship and accreditation thereof.***

The author Mr. /Ms. Marta Bravo Lázaro

**HEREBY DECLARES** that he/she owns the intellectual property rights regarding the piece of work: Software Integration and Simulation of Autonomous Vehicles that this is an original piece of work, and that he/she holds the status of author, in the sense granted by the Intellectual Property Law.

## ***2. Subject matter and purpose of this assignment.***

With the aim of disseminating the aforementioned piece of work as widely as possible using the University's Institutional Repository the author hereby **GRANTS** Comillas Pontifical University, on a royalty-free and non-exclusive basis, for the maximum legal term and with universal scope, the digitization, archiving, reproduction, distribution and public communication rights, including the right to make it electronically available, as described in the Intellectual Property Law. Transformation rights are assigned solely for the purposes described in a) of the following section.

## ***3. Transfer and access terms***

Without prejudice to the ownership of the work, which remains with its author, the transfer of rights covered by this license enables:

- a) Transform it in order to adapt it to any technology suitable for sharing it online, as well as including metadata to register the piece of work and include "watermarks" or any other security or protection system.
- b) Reproduce it in any digital medium in order to be included on an electronic database, including the right to reproduce and store the work on servers for the purposes of guaranteeing its security, maintaining it and preserving its format.
- c) Communicate it, by default, by means of an institutional open archive, which has open and cost-free online access.
- d) Any other way of access (restricted, embargoed, closed) shall be explicitly requested and requires that good cause be demonstrated.
- e) Assign these pieces of work a Creative Commons license by default.
- f) Assign these pieces of work a **HANDLE** (*persistent URL*). by default.

## ***4. Copyright.***

The author, as the owner of a piece of work, has the right to:

- a) Have his/her name clearly identified by the University as the author
- b) Communicate and publish the work in the version assigned and in other subsequent versions using any medium.
- c) Request that the work be withdrawn from the repository for just cause.
- d) Receive reliable communication of any claims third parties may make in relation to the work and, in particular, any claims relating to its intellectual property rights.

## ***5. Duties of the author.***

The author agrees to:

- a) Guarantee that the commitment undertaken by means of this official document does not infringe any third party rights, regardless of whether they relate to industrial or intellectual property or any other type.



- b) Guarantee that the content of the work does not infringe any third party honor, privacy or image rights.
- c) Take responsibility for all claims and liability, including compensation for any damages, which may be brought against the University by third parties who believe that their rights and interests have been infringed by the assignment.
- d) Take responsibility in the event that the institutions are found guilty of a rights infringement regarding the work subject to assignment.

**6. Institutional Repository purposes and functioning.**

The work shall be made available to the users so that they may use it in a fair and respectful way with regards to the copyright, according to the allowances given in the relevant legislation, and for study or research purposes, or any other legal use. With this aim in mind, the University undertakes the following duties and reserves the following powers:

- a) The University shall inform the archive users of the permitted uses; however, it shall not guarantee or take any responsibility for any other subsequent ways the work may be used by users, which are non-compliant with the legislation in force. Any subsequent use, beyond private copying, shall require the source to be cited and authorship to be recognized, as well as the guarantee not to use it to gain commercial profit or carry out any derivative works.
- b) The University shall not review the content of the works, which shall at all times fall under the exclusive responsibility of the author and it shall not be obligated to take part in lawsuits on behalf of the author in the event of any infringement of intellectual property rights deriving from storing and archiving the works. The author hereby waives any claim against the University due to any way the users may use the works that is not in keeping with the legislation in force.
- c) The University shall adopt the necessary measures to safeguard the work in the future.
- d) The University reserves the right to withdraw the work, after notifying the author, in sufficiently justified cases, or in the event of third party claims.

Madrid, on 29..... of June 2018.....

**HEREBY ACCEPTS**

Signed... María Bravo .....

Reasons for requesting the restricted, closed or embargoed access to the work in the Institution's Repository



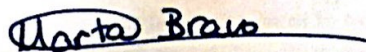


Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título  
SOFTWARE INTEGRATION AND SIMULATION OF AUTONOMOUS  
VEHICLES

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en  
el

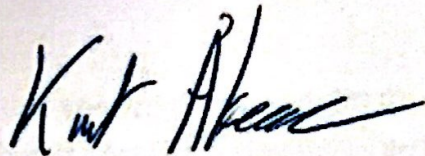
curso académico 2017/2018 es de mi autoría, original e inédito y  
no ha sido presentado con anterioridad a otros efectos. El Proyecto no es  
plagio de otro, ni total ni parcialmente y la información que ha sido tomada  
de otros documentos está debidamente referenciada.

Fdo.: Marta Bravo Fecha: 29/ 06/ 2018



Autorizada la entrega del proyecto  
EL DIRECTOR DEL PROYECTO

Fdo.: Knut Åkesson Fecha: 29/ 06/ 2018







ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA  
(ICAI)

GRADO EN INGENIERÍA ELECTROMECAÁNICA

Especialidad Electrónica

# **SOFTWARE INTEGRATION AND SIMULATION OF AUTONOMOUS VEHICLES**

Author: Marta Bravo Lázaro

Director: Knut Åkesson

Madrid

July 2018



## Acknowledgements

Firstly, I would like to thank all my team mates Anton Frigård, Erik Lund, Oscar Harmsen and Svante Treismo for all the collaboration and all the hours we have spend together to get this project done. All the project was done in collaboration with them as well as the report [30]. Also to Ashfaq Hussain Faarooqui for the guidance and feedback throughout the project. I am also thankful to Knut Åkesson for putting this project together and enable me to work with such an ambitious project eventhough I was not from a chalmers bachelor.

I would also like to thank all the groups who have been working on the Autonomous Twizy, especially the perception and safety group for helping us. And to Chalmers University of Technology for giving me the chance of being involved in such a big and important project. And last but not least the Apollo team and community.

Marta Bravo

Madrid, July 2018

# SOFTWARE INTEGRATION AND SIMULATION OF AUTONOMOUS VEHICLES

**Autora: Bravo Lázaro, Marta**

Director: Knut Åkesson

ICAI, Universidad Pontificia de Comillas

## RESUMEN DEL PROYECTO

El presente proyecto que trata sobre la creación de un entorno virtual es una parte de un proyecto grupal cuyo objetivo es la automatización de vehículos. En este proyecto, en concreto, se ha pretendido crear y analizar un entorno a través de diferentes situaciones o escenarios habituales. Para ello se contó con Apollo Auto 2.0. software abierto que posibilitó el desarrollo de este proyecto. Más concretamente se trabajó principalmente con el módulo de simulación.

**Palabras clave: Apollo Auto, ROS, Autonomous vehicle, AVs, Simulation, Virtual Testing.**

### I. Introducción

Actualmente el mundo virtual y de las simulaciones se encuentra en auge en campos como la fabricación o la producción. Se considera una manera eficiente de analizar y concluir la viabilidad de los proyectos, además de un método de ahorro para la economía de la empresa al no tener la necesidad de construir y probar prototipos.

Por otro lado, dentro de la industria de la automoción la conducción autónoma es uno de los sectores con más empuje e investigación. Se considera como una forma de reducir significativamente el número de accidentes consiguiendo carreteras más eficientes y seguras.

El presente proyecto trata sobre el estudio y análisis de un ambiente virtual recreado a través de la plataforma Apollo Auto. Dicho entorno está creado para analizar el comportamiento de un vehículo autónomo

ante diferentes escenarios que recrean situaciones de tráfico comunes.

### II. Objetivos

El objetivo global del proyecto consiste en conseguir automatizar un Renault Twizy.

Los objetivos concretos que se deseaban alcanzar a lo largo de este proyecto en particular son:

- Aprender a usar Apollo Auto 2.0. y ROS
- Crear un simulador
  - Crear un mapa
  - Implementación de obstáculos, tanto estáticos como dinámicos
- Diseño de escenarios
- Evaluación y análisis de los escenarios

### III. Metodología

Al comienzo se planteó el proyecto como la base para integrar los códigos de los diferentes grupos del proyecto global. Códigos como el de seguridad o el que modela el vehículo con los parámetros adecuados. Finalmente, se reorientó, centrándolo en la creación de un entorno de simulación para poder probar los diferentes códigos y el comportamiento del vehículo antes de ponerlo en práctica.

El primer paso que fue considerado y realizado al inicio fue el reunir información acerca de Apollo Auto. Debido a la falta de información online decidimos crear un documento, el “how to” adjuntado en los apéndices con todo aquello que se fuese reuniendo. Esta ha sido una forma sencilla de compartir la información, procedimientos y comandos. Además de servir como herramienta para la posterior

ampliación de este proyecto en años futuros.

Desde un punto de vista más práctico, se comenzó con el análisis de los bagfiles. Estos son grabaciones de diferentes situaciones que venían ya en Apollo Auto. Fueron usadas para analizar el output de módulos como el mapa, Dreamview (simulación), Perception, Routing o Planning, de forma que pudiéramos tener una referencia inicial sobre lo que deberíamos esperar como output en nuestros escenarios. Sin embargo, la poca flexibilidad que tienen al ser grabaciones y no dejar al usuario elegir el escenario que desea simular hace que resulte un método inútil hasta cierto punto. Esto fuerza a generar un código que haga que los obstáculos y el mapa sean personalizables y que viene detallada su documentación en el capítulo 4.

Posteriormente se comenzó con el desarrollo del mapa y los obstáculos. Estos se crearon de manera más o menos simultánea. Para finalizar con la simulación de los diferentes escenarios, cambiando así los parámetros del mapa y los obstáculos creados por nosotros mismos en el paso previo. En concreto este proyecto se centró en el desarrollo de los obstáculos tanto estáticos como dinámicos. Por otro lado, tanto la creación como la evaluación de los distintos escenarios. Para crear los escenarios se ha necesitado entender los archivos de los mapas ya que necesitaban crearse archivos binarios para poder reproducirlos en Dreamview.

Los otros miembros por un lado trabajaron con la parte de integración y comunicación con otros grupos. También se encargaron de todo lo relacionado con el sistema operativo dentro del Twizy, estanterías dentro del vehículo, instalación y configuraciones.

En cuanto al uso de dispositivos, en mi caso concreto, comencé utilizando mi ordenador personal un ASUS a través de una máquina virtual. Finalmente, debido a un bucle que no supe identificar y ocupaba prácticamente la totalidad de la memoria del ordenador se cambió al ordenador proporcionado por La Universidad Tecnológica de Chalmers y

que viene explicado en detalle en el capítulo 2.

#### IV. Apollo Auto

Se trata del software empleado para desarrollar el proyecto. Está construido sobre ROS que a su vez está construido sobre Ubuntu. Se trata de un software libre, en el cual los programas están divididos en módulos o nodos. Estos se conectan para mandar mensajes entre ellos a través de topics como publicadores o suscriptores. El gran punto de este tipo de software es la flexibilidad que tiene al poder activar y modificar ciertos módulos sin necesidad de usar o modificar todos simultáneamente. Será una cualidad muy usada a lo largo del proyecto. En nuestro caso se modificarán módulos como el mapa o el de simulación, también llamado Dreamview.

#### V. Resultados

La forma que se ha tenido para evaluar el entorno virtual es analizando si los símbolos preestablecidos por Apollo aparecen en pantalla. No se puede olvidar que el objetivo es el análisis y la evaluación del ambiente y no del comportamiento del vehículo.



Ilustración 1. Algunos símbolos de Dreamview

En el caso del peatón cruzando sobre el paso de cebra, esperaríamos ver, por un lado, que el obstáculo identificado por el coche fuera un peatón, ya que hay un símbolo específico (b) que significa que un peatón ha sido identificado. Sin embargo, lo que aparece es un cono naranja, esto debe interpretarse como que obstáculo cualquiera ha sido detectado. Se considera que el posible motivo es la falta de información en el protobuf message. Por otro lado, el símbolo del paso de cebra (a) tampoco aparece, en este caso podría estar bien implementado. Esta conclusión viene motivada por el análisis en uno de los mapas que ya venían por defecto en los archivos de Apollo Auto. En este mapa

tampoco se consiguió que apareciera el símbolo que hacía ver que el vehículo había identificado el paso de cebra.

En el segundo escenario, en el cual se pretende que se siga al vehículo de delante ocurre el mismo problema. El cuadro de “seguir a otro vehículo” es sustituido por el cuadro de “parar”. Se deduce que el error es el mismo que en el escenario anterior. La falta de información del protobuf message.

Por último, en el caso de cambiar de carril, tras varias simulaciones se concluye que no siempre se consigue el cambio de carril. Esto puede ser porque no se resetea adecuadamente toda la información de todos los módulos usados previamente. Por otro lado, el símbolo de cambio de carril no aparece en pantalla. Sin embargo, al probarlo en mapas originales de Apollo 2.0. tampoco aparece. Por tanto, se concluye que el cambio de carril se produce satisfactoriamente.

Pese a los errores descritos previamente sabemos que el tipo de obstáculo no es elegido al azar por el programa, sino que responde a lo que el usuario pretende simular. Esto se puede confirmar gracias al color que se presenta en Dreamview y que se corresponde con el que se marca en el manual de Apollo. Un ejemplo sería el peatón que está representado con un rectángulo amarillo tal y como era esperado.

## **VI. Conclusión**

La principal conclusión es la correcta implementación del ambiente de simulación. Es decir, el entorno creado posibilita el análisis del comportamiento del vehículo que es el objetivo de toda simulación. No se puede olvidar la falta de ciertos símbolos, todo esto ha sido explicado con más detenimiento en el apartado de resultados.





# SOFTWARE INTEGRATION AND SIMULATION OF AUTONOMOUS VEHICLES

**Autora: Bravo Lázaro, Marta**

Director: Knut Åkesson

ICAI, Universidad Pontificia de Comillas

## PROJECT SUMMARY

**The projects which aim is the creation of a virtual environment is a section of a collective project which end is the vehicle automatization. The core of this particular project was the analysis and creation of common driving scenarios. For this aim an open software platform, Apollo Auto 2.0. was used. In particular the module that was mainly used was the simulation module.**

**Key words: Apollo Auto, ROS, Autonomous vehicles, Twizy, Simulation, Virtual Testing.**

### I. Introduction

Virtual world and simulations are currently seen as growing sectors in some fields such as production and manufacturing. It is considered as an efficient way of analyzing and concluding the feasibility of some projects. Moreover, it is a method that saves vast amounts of money since no prototype needs to be created nor tested.

On the other hand, the automation industry is investing a lot in the research of autonomous driving. This is motivated by the decrease of accidents. Therefore, we could conclude that the autonomy derives in more efficient and secure roads.

The present project is related with the analysis of a virtual environment done thank you to Apollo Auto. This environment is made with the end of testing the behavior of the autonomous vehicle in different scenarios. These situations pretend to recreate common traffic scenarios.

### II. Goals

The global goal of the Project is to achieve an autonomous Renault Twizy driving through some of the campus streets at Chalmers University.

More specifically, the goals of this particular thesis are:

- Learning of Apollo Auto 2.0. and ROS
- Create the simulation environment
  - Customize a map
  - Obstacle implementation of both, dynamic and static obstacles.
- Different scenario simulation
- Analysis and evaluation of the virtual environment.

### III. Methodology

The idea of this project at the beginning was to join the codes of the different groups in Apollo so these codes could work all together. However, the project finally was reoriented. The main goal was the creation of a custom environment to check the modifications made by the different groups before testing them in the actual Twizy.

First step considered was the gathering of information about Apollo Auto. The lack of information about Apollo Auto forced us to start the “how to”. It is a document seen as an easy way of sharing information, procedures or commands. It was created by all members of the group. Moreover, it can also be used as a learning tool for future developments of the project. This document is attached in the appendix of the project.

From a more practical prospective, we started analyzing the bagfiles. This are recordings of different situations that came

along with Apollo Auto. They were used for analyzing the output of modules such as map, Dreamview (simulation module), Perception, Routing or Planning. This enabled us to have an initial reference of what to expect as outputs of our own scenarios. However, the little flexibility that the bagfiles have by not letting the user to choose which scenario to play, makes this method kind of useless. This ends up on the need of coding to create customizable maps and obstacles. All this documentation is further explained in chapter 4.

Afterwards both, the map and the obstacles, were designed and codified. This task was made simultaneously. It was done to end playing different scenarios, enabling the user to customize the scenario by changing the values of the parameters in the map or the obstacles that were created in the step before. More precisely, this project was focus on the develop of static and dynamic obstacles. Moreover, the creation and the evaluation of the virtual environment are also in the scope of the project. In order to be able to create the scenarios, maps needed to be understood to enable the creation of binary files. This binary files were necessary to display the map in Dreamview.

The other members of the group were focused on the integration and communication with other groups. Besides, the installation of the onboard computer was also made by them, the computer rack for the Twizy, and the installation and configuration of the onboard computer.

About the devices that were used during the project, in my case, my personal computer was used at the beginning, an ASUS by a Virtual Machine that allowed me to run Ubuntu. Finally, due to a loop that could not be identified made me start working with the computer provided by Chalmers University of Technology. This is explained more in more detail in chapter 2.

#### IV. Apollo Auto

Apollo Auto is the software used to develop the whole thesis. It is built upon ROS which at the same time is built upon Ubuntu. It is an open source platform in which the programs are divided in modules or nodes.

The connection among them is by topics with a publisher/subscriber method. The main attribute of this system is the isolation of the modules, in other words, the way of activating or modifying some of the modules and not all at the same time. This was used during the project. In this project some modules such as the map or the simulation module, also called Dreamview, will be modified.

#### V. Results

The evaluation method of the virtual environment is based on the analysis of the symbols that are pre-established by Apollo. We cannot forget that the goal here is to evaluate the environment and not the behavior of the vehicle.



Illustration 2. Some Dreamview symbols

In the scenario of a pedestrian crossing the road, we expected the vehicle to notice the presence of a pedestrian. Therefore, it was assumed that the pedestrian symbol (b) should have appear. However, an orange cone did appear instead. Thus, we know that a random obstacle has been noticed by the Apollo vehicle. It is considered that this could be because some information is missing in the protobuf message. We also have the crosswalk symbol (a) which does not appear either. Nevertheless, it could have been correctly implemented. This conclusion is driven by the analysis of maps that already came by default in Apollos files. In these maps the expected crosswalk symbol did not appear either.

In the second scenario, following a vehicle the same problem is encountered. The assuming frame that might have appear, the “following another vehicle” frame is changed by a “stopping” frame. We deduce that the error might be the same as before, the lack of information in the protobuf message.

Finally, the change of lane scenario. After several runs, it was concluded that the change of lane was not fulfilled every time. This issue might be because of some data not being reset properly. Moreover, the symbol is not shown. However, it was tested in some original Apollo file and it was not shown either. We conclude that the lane switching was properly implemented.

Even though, we have described some errors, we also know that the program recognizes the obstacle that the user has implemented. This can be said because of the color that the obstacle box presents in Dreamview that matches up with Apollo specifications. i.e. the pedestrian is yellow as it was expected.

## **VI. Conclusion**

The main conclusion is that the virtual environment was properly implemented. This conclusion is based on the fact that the environment created let the user analyzed the vehicle behavior which is the goal of any simulation. However, we cannot forget the absence of some symbols. This is explained in more detailed in the evaluation and results sections.





ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA  
(ICAI)

GRADO EN INGENIERÍA ELECTROMECÁNICA

Especialidad Electrónica

**MEMORY**

**SOFTWARE INTEGRATION AND  
SIMULATION OF AUTONOMOUS  
VEHICLES**

Author: Marta Bravo Lázaro

Director: Knut Åkesson

Madrid

July 2018

# Memory Index

<b>Part I Memory .....</b>	<b>.....</b>
<b>Chapter 1 INTRODUCTION .....</b>	<b>11</b>
1.1 State of art.....	12
1.2 Motivation.....	14
1.3 Aim .....	15
1.4 Method.....	16
1.5 Resources .....	18
1.6 Hierarchy.....	18
<b>Chapter 2 SOFTWARE &amp; HARDWARE .....</b>	<b>23</b>
2.1. Software .....	23
2.1.1. ROS.....	23
2.1.1.1. Master.....	24
2.1.1.2. ROS commands.....	25
2.1.2. APOLLO .....	26
2.1.2.1. Modules.....	26
2.1.2.2. Communication within Apollo .....	30
2.1.2.3. Adapters .....	30
2.2. Hardware.....	31
<b>Chapter 3 Simulation Module .....</b>	<b>33</b>
3.1. Dreamview system notification scheme.....	35
<b>Chapter 4 Creating and testing a simulation environment .....</b>	<b>37</b>
4.1. Simulation using bag files .....	38
4.2. Creating the environment.....	42
4.2.1. Creating a road.....	42
4.2.2. Obstacle implementation .....	46
4.2.3. Simulation methods advantages and disadvantages.....	48
4.3. Complete system simulation .....	50
4.3.1. Scenario 1: Pedestrian crossing the road.....	50
4.3.2. Scenario 2: Following another car .....	55
4.3.3. Scenario 3: Change of lane.....	57
4.3.4. Scenario 4: Avoiding static obstacles.....	<b>¡Error! Marcador no definido.</b>

4.4.	Conclusions and general discussion .....	59
<b>Chapter 7</b>	<b>ENVIRONMENTAL IMPACT .....</b>	<b>61</b>
<b>Chapter 9</b>	<b>CONCLUSION .....</b>	<b>63</b>
<b>Chapter 10</b>	<b>FUTURE DEVELOPMENTS .....</b>	<b>65</b>
<b>Chapter 11</b>	<b>BIBLIOGRAPHY .....</b>	<b>67</b>
<b>Part II Budget_</b>	<b>.....</b>	<b>67</b>
<b>Chapter 1</b>	<b>MEASUREMENTS .....</b>	<b>73</b>
1.1	Hardware.....	73
1.2.	Software .....	73
1.3.	Labor costs .....	73
<b>Chapter 2</b>	<b>UNIT PRICE .....</b>	<b>75</b>
1.1	Hardware.....	75
2.2.	Software .....	75
2.3.	Labor costs .....	75
<b>Chapter 3</b>	<b>PARTIAL AMOUNT .....</b>	<b>77</b>
3.1.	Hardware.....	77
3.2.	Software .....	77
3.3.	Labor costs .....	77
<b>Chapter 4</b>	<b>TOTAL AMOUNT .....</b>	<b>79</b>





## Figure Index

Figure 1. Gantt Chart.....	17
Figure 2.Related Research Areas .....	21
Figure 3. Flow chart .....	21
Figure 4. Publisher and subscriber initiation process .....	24
Figure 5. IPC used.....	32
Figure 6. Dreamview interface.....	34
Figure 7. Some of the symbols representing the decision making made by the system in the Dreamview visual notification scheme.....	35
Figure 8. Symbols representing what causes the car to stop in the Dreamview visual notification scheme.....	35
Figure 9. The structure of the virtual wo-lane road used in the simulations. ....	42
Figure 10. Straight Lane in Dreamview .....	45
Figure 11. Four important moments in time from the scenario of the crossing pedestrian.....	51
Figure 12. Four important moments in time from the scenario chasing another car .....	56



## Table index

Table 1. Apollo publishing & subscribing structure .....	29
Table 2. Subscriptions and publications of Dreamview .....	34
Table 3. A printout of information about the demo_2.0.bag file. ....	39
Table 4. An output message from the prediction module during simulation using bagfiles.....	40
Table 5. The output after generating the routing and sim maps.....	45
Table 6. Output message from the planning module when stopping dor the crossing pedestrian ...	52
Table 7. Parts of a routing response message from the routing module after sending a routing request to switch lanes. ....	57





# ***PART I MEMORY***



## Chapter 1 INTRODUCTION

The number of vehicle manufacturers that are getting involved with autonomous vehicles is rapidly increasing. This increase has led to a fierce competition when it comes to delivering safe, usable and reliant autonomous cars. Nowadays, manufacturers have the need of quicker and more effective ways to test the quality of their algorithms/systems in order to ensure their needs are met. To secure that the algorithms are correct and safe they must be tested for various scenarios to make sure they are safe for consumer use. Testing the algorithms in the real world is expensive, time consuming and unsafe. Moreover, not all scenarios can be tested in the real world. By simulating real world scenarios, the algorithms can be tested. Hence, simulations are vastly important for developing safe and reliable autonomous vehicles, hereinafter referred to as AV.

To ensure a car can autonomously drive, several problems need to be solved, such as perception, path planning, vehicle control and safety. Once these sub-systems are implemented and tested separately, they must be integrated and tested together. Testing consists of simulating different scenarios to ensure reliable and safe performances.

In this project we aim to autonomously drive a Renault Twizy. In order to do this, the project is divided into several groups each focusing on one sub-system. To simplify the problems that we will need to deal with, a platform called Apollo Auto will be used for integration as well as testing. The platform, Apollo Auto (The Apollo Team 2018), is built on the Robot Operating System (ROS) and utilizes the built-in functions in ROS to support development of AVs. These functions in ROS and Apollo facilitates the development and make this project possible in this timeline.

---

## 1.1 *State of art*

---

The world of autonomous driving is yet unexplored and in research. There are a lot of companies doing their best to make that futuristic goal become real. It is worthy to explain how are defined the different autonomy levels. A definition of autonomy in vehicles is given by SAE (Society of Autonomous Engineers), where 5 levels are described. Levels 1 and 2 describe aids for the driver such as adaptive cruise control. Levels 3-5 define an increased amount of self-driving capabilities in a vehicle [1]. Examples of autonomous vehicles include; robotic lawn mowers, cars with lane-detection or driverless buses [2]. The definition is broad and incorporates a varied range of machines. In this report however, the focus will lie on cars.

Google, in 2009 a new Google's self-driving car project started, Waymo. They have invested more than 1 billion dollars into the AV research [3]. Waymo currently have a fleet of cars that achieve to drive more than 25,000 autonomous miles per week [4]. They have already pointed out that they have let some driverless cars in some parts of the United States.

According to Elon Musk, Tesla's founder, cars will drive by themselves in two years. From 2014 onwards, this company has been researching and testing with autonomous cars. They have been ahead in this sector for many years achieving a level 3 autonomy in their vehicles [5].

The Drive Me project at Volvo is being implemented in Gothenburg, Sweden. It is the world's biggest large-scale pilot project in autonomous driving. It is a project which is planned to be extended to London and China afterwards. The end of this project is to receive some feedback from their customers with the implementation of level 4 autonomy vehicles. (No need of a human driver but with a steering wheel and pedals in case of emergency). Drive Me last announcements have stated that 100 people will be involved in the program within the next four years [6].

Moreover, some big companies like Uber, General Motors, Delphi or BMW among others, are now trying to reach level 4.

Another perspective would be the software development and implementation. At the moment, NVIDIA's CEO, Jen-Hsun wants his robo-car computer package to become

an industry standard. Nvidia has already released an automotive package, called Drive PX. He stated that there is a collaboration between Nvidia and Audi. They are planning to present a Level 4 vehicle as well [7].

Zenuity is a new company that is working with Volvo and Autoliv. They are planning to use Nvidia's AI computer groundwork as a basis for their own software. This software is exclusively for Volvo [8].

---

## 1.2 *Motivation*

---

The transportation of both, commodities and people, has been a need that the human being have always have. Some significant developments have been made over the years in the transportation sector. First of all, the invention of the wheel, afterwards the carriages to end up with the vehicles that we know nowadays. The main point of this bachelor thesis is to get deeper into a new way of thinking about transportation.

Nowadays there is a revolution in the automation sector. The increase in the number of vehicle manufacturers and the research that companies are making witnesses it. Some key factors that motivate this research are the statistics around car accidents. More than 1.25 million people die each year as a result of road traffic crashes [9]. Another key factor is the big change in the way of thinking due to the new technologies. This has led into a world where everything needs to be connected. The sustainability, safety and effectiveness are points that are also considered. The implementation of AVs would bring safer and more sustainable roads. Since some traffic congestions would be alleviated a lot of fuel would be saved. Because of all these mentioned points, a need of reinventing the automation has emerged.

In my personal case, I came to Gothenburg to finish my degree, specifically to Chalmers University. This university has got some good collaborations with Volvo Group. Hence, they have a vast knowledge about vehicles and transportation. Furthermore, Volvo is now developing one of the biggest projects of autonomous cars in Gothenburg. Therefore, I thought about taking advantage of this situation and get some deep knowledge about autonomous driving. On the other hand, this project was offered to be made in groups with Swedish people. That has given me a lot of work experience. In fact, one of the main challenges that we had to face was the communication, planning and working, together with the other groups, such that each group would have the possibility of fulfilling their respective goals as well as collectively trying to reach the project goal. To conclude this first introductory part is in collaboration with them. []



---

## 1.3 *Aim*

---

This thesis will be focused on the software integration and testing using the Apollo-framework as well as hardware implementation of the onboard computer.

From a simulation or testing standpoint, a simulation needs to be done. This will ensure the proper performance of the Twizy. This recreation will need different scenarios around the building area. We could divide this goal into different sub-tasks such as the design of a map and the implementation of some obstacles into that map. The obstacles implemented should be both, dynamic and static. According to the place where the Twizy is intended to run, the map that needs to be simulated is one of Chalmers buildings.

The software part of this thesis will be the joint of the different modules, some of them were already coded in Apollo. However, some of the groups have made their own codes. Therefore, all codes need to work together properly. The aim is to simulate the complete system of the Twizys software (perception, path-planning and control, safety system). It could be said that this will be done by using knowledge of how Apollo works and establishing how the software architecture should be through the collaboration with the other groups.

Finally, the hardware work will be the implementation of the onboard computer. For that aim a rack will be design and assembled inside the car. Furthermore, that onboard computer will need to be set up so that Apollo can be run.

As a summary we could say:

- Simulation world/Testing
  - Map
    - Straight roads
    - Curves
  - Objects
    - Static
    - Dynamic
- Software → Modules
- Hardware/Integration
  - Set up onboard computer
    - Rack
    - Computer set up

---

## *1.4 Method*

---

On the basis that the bachelor thesis was made by five people, for efficiency reasons, the work has been split into different sections. This thesis will be about the simulation of obstacles and map creation.

As an overall view it can be said that the obstacles work will start with the implementation of static objects into the simulations. Nevertheless, dynamic obstacles such as bicycles, pedestrians or other vehicles will intend to be done as well. On the other hand, some maps are needed. The goal is to achieve some maps of the building of Chalmers.

For Apollo to be an effective tool in achieving the goal, the project will begin with learning the structure of the platform. The learning goal is to have a good knowledge of how to build a software system reliably and effectively, model the intended route as well as creating real world scenarios. Even though the work is split each group member must have a basic understanding of the work of his or hers colleagues and time will therefore be allocated for this.

The work in itself will be performed on the group members personal laptops as well as remotely on stationary computers. One of these computers is situated in the examiner's office and equipped with a powerful graphics card. The other computers, equipped with great processing power but no advanced graphics card, are situated in a special lab. Most of the work was done in my personal laptop where a VirtualBox was installed. Therefore, Linux could be run and consequently Apollo/ROS as well.

To be able to create a virtual model of the maps some data might be obtained through testing. For the environment an integrated map creator tool exists in Apollo and will be used. This tool uses input data from sensors on the car to create the environment such that it is available for use in simulations.

Parallel with the modelling, work on creating random scenarios will be done. This will include making these scenarios appear automatically and in a randomized order. When the automatic scenarios are done the simulation can run for hours which will be great for detecting possible issues with the algorithms.

When there is a simple model and a functioning simulator building a steel frame in the back of the car will be the next priority. This frame will house the onboard computer and possibly separate hardware, running safety algorithms. There might also have to be space for a secondary battery for powering the computer and sensors. This will be the case if the battery that is currently in the Twizy and is powering the motor cannot be used for this purpose as well.

All the work will be done in parallel with the work made in other groups. There exist dependencies and we would need to collaborate with the different teams. The integration team will initially work with Docker Containers and ROS objects, e.g. creating images, containers, nodes, topics and packages etc. Early tests will initiate basic communication between different nodes. In these cases, the data sent might not be same type of data sent in latter stages of the project. Besides, information regarding actual input and output from the subsystems will therefore be gathered through conversation with the other project groups. Fake data will then be obtained and tested with the system. By the time the Twizy is running for data gathering, the real-world data will be collected and used in further testing. The final stage will be to integrate the Apollo framework on the onboard computer. This will include installing Ubuntu and Apollo Auto on the computer as well as making sure all the subsystems are functioning properly. Continuous testing on the personal workstations as well as on the onboard computer will provide a base for improving the system.

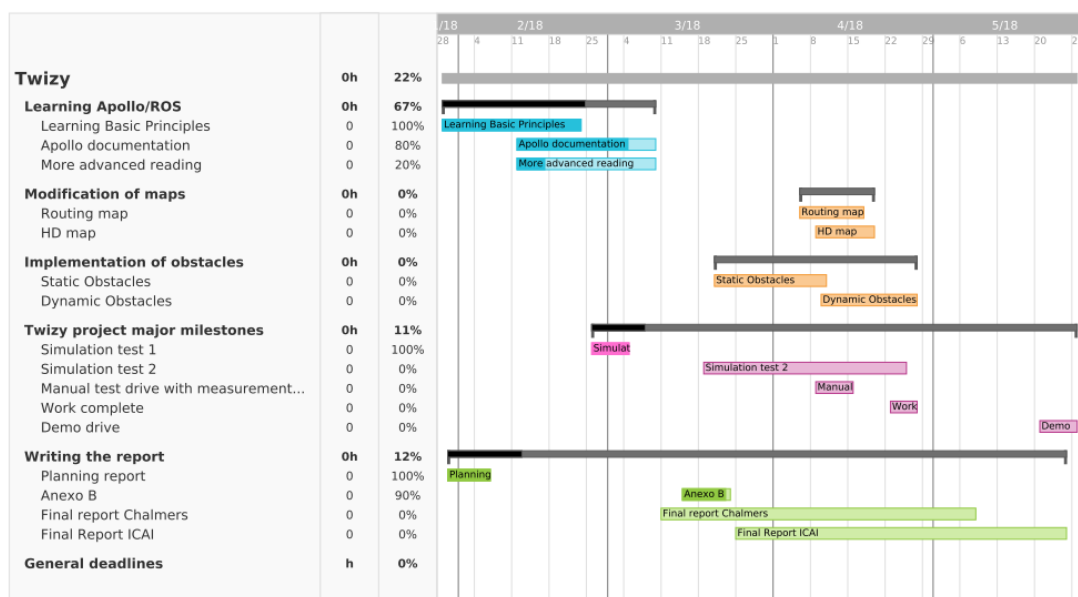


Figure 1. Gantt Chart

---

## *1.5 Resources*

---

The main resource that will be used is Apollo. It is an open autonomous driving platform made by Baidu a Chinese company [10]. This platform is made for developing autonomous driving algorithms. Apollo is built upon a software library called Robot Operating System (ROS), which is also an open source. ROS has a great library for development regarding robots and automated machines.

The history of Apollo starts in 2015 when the Chinese company Baidu was decided to invest in the research of AVs having driving tests on the highways and urban roads of Beijing. In 2016 they obtained a license that let them test the self-driving vehicles in California [11]. The first software released Apollo 1.0. was first released to everyone in 2017. Finally, in 2018 was launched in Las Vegas, Apollo 2.0 that will be used during the entire project [12][31]. A latest version of Apollo is now being launched, Apollo 2.5. The use of this version will not concern this thesis.

---

## *1.6 Hierarchy*

---

The overall aim of this project was to make the Twizy drive autonomously through an established path in Chalmers. We were around thirty Chalmers students involved in the project and thus groups needed to be made. Seven different groups of about 6 people each were created. This division had the end of making the work more efficient by having some “specialist” in each of the tasks that were considered essential. At the end, all the groups were related and had to work cooperatively.

This section was open to explain the different departments that were created and to describe in a general way the tasks that needed to be done by each of them.

### Inductive charging:

It was decided that the Twizy should be charged by itself with the end of making the vehicle more ecofriendly and with more km of autonomy. Therefore, the induction team needed to create a prototype which made the charges of the vehicle via induction

possible. They were focused on the coils and Twizys equipment. The design should bear the outdoor climate of Gothenburg and be supplied from a stationary battery supply [13].

#### Power modelling for energy optimization:

Energy should be optimized by selecting different routes, speeds and ways of speed up or decelerate due to the expected arrival time. This teams aim was to evaluate the different operation strategies in power and energy. The main idea was to design a general power model and then adapt that model to the Twizy measurements [14].

#### Steering and Motor control

All the work was divided in two sub-teams. One group did the emergency stop and the steering of the car. The car was at first tested remotely, they oversaw the remote control.

Second group was more focused on power electronics. They had been working in converting the Twizy from pure mechanical steering into autonomous steering. This was done by adding electric power steering. This EPS is controlled by a programmable motor control unit. They needed a special collaboration with others such as safety or vehicle control. They needed to ensure that messages from the vehicle control group were correctly processed and done [15].

#### Safety systems

Their responsibility was to protect both driver and environment from harm. To that end some algorithms had to be done so the vehicle can react when an obstacle is detected. Once an object is detected an action should be taken by the car according to what had been detected. The safety system implemented was “hard” which means that the emergency breaks is activated whenever the vehicle determines a possible or imminent danger. They must ensure that every action that is delivered through the bus CAN is listened by steering and motor control [16].

#### Path Planning and vehicle control

The aim of this project was to design and experimentally validate a path planner and a low-level vehicle control system. The path planner had to be able to decide a path based on the obstacles of the surrounding environment. The car should have had a control implemented in order to follow the path that was decided. The low-level control system should handle this task. Hence, it commands steering and acceleration/braking to follow

the path that was determined by path planning. The work was divided in two sub-teams as in steering and motor control. Path planning which decided the safest and best route and vehicle control that processed the information from path planning.

Vehicle control was situated between perception and safety. Their input is all the data regarding the state of the vehicle and the perceived world around it. On the other hand, their output data is originally a throttle and a brake command in percent of full throttle and brake. A steering target is also another output. All this information needs to be validated by safety, so the vehicle can conclude that it is safe. Finally, some of the specific parameters in the code to fit the Twizy instead of the Lincoln MKZ will be modified [17].

### Perception

The aim of the project was to help the Twizy knowing what is around. This team was involved with the installation of sensors such as, GPS, cameras, radar, sensors and laser scanners (lidars) in the autonomous Twizy and the corresponding algorithms to utilize the sensor output data in an efficient and reliable way.

This group needed to work together with groups like path planning and control to develop an estimation algorithm that takes a coherent view of the environment with the sensor observations [18].

### Virtual Integration and Testing

This bachelor project will be focused on this section. As an overall overview the aim of this project is to create a virtual environment, integrating the different modules and testing the Twizy. A virtual environment on top of ROS will be built. A coordination of the work will need to be done so all the components can work together and tested in the different simulations. Besides, obstacles, maps and a virtual model of the car will be done [19].

<b>Visual navigation and perception</b>	<b>Sensor fusion and object tracking; Wireless communication</b>	<b>Decision making and control</b>	<b>Energy storage and electric drivetrain</b>
Computer Vision (Fredrik Kahl)	Sensor fusion and tracking (Lars Hammarstrand) Localisation and communication (Henk Wymeersch)	Trajectory generation and following (Paolo Falcone)	Energy storage, inductive charging, regenerative braking (Torbjörn Thiringer, Yuing Liu)
<b>Safety, verification and testing (Knut Åkesson, Martin Fabian, Petter Falkman)</b>			

Figure 2. Related Research Areas

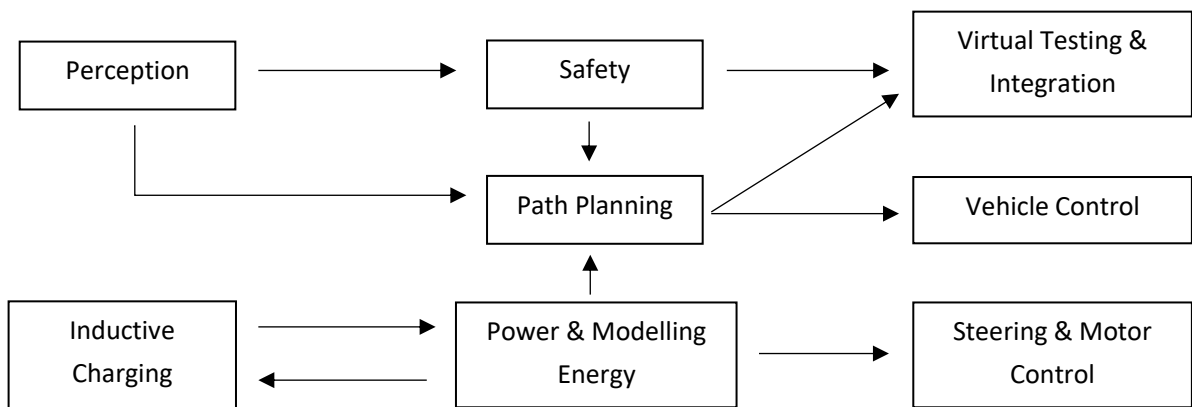


Figure 3. Flow chart





## Chapter 2 SOFTWARE & HARDWARE

In this chapter the ROS-library and foundations of Apollo Auto are described. It is meant to be an introduction to some basic concepts of ROS and Apollo Auto to enable the reader to begin working on their own with AVs. A hardware section has been written as well so the on-board computer installed in the Twizy can be analyzed.

---

### 2.1. *Software*

---

Apollo Auto is an open sourced platform for developing autonomous driving algorithms. It is developed by a Chinese company called Baidu. Apollo is built upon a software library called Robot Operating System (ROS) which is also open source.

#### 2.1.1. *ROS*

---

ROS is a globally used framework which main goal is the develop and research in robotics. However, ROS is built upon Linux or Ubuntu. Because of this, ROS is also known as a meta-operating system.

The basis of ROS is grounded in the way communication is achieved within a complete robotics system [20]. As most robotic systems are constructed of several sub-systems (nodes), which need to communicate, a vital part of problem solving is finding solutions so that they can talk to each other.

In ROS communication is achieved by using what is called nodes and topics. Nodes are the actual programs that operate sensors, motors, etc. They are responsible for the computation that is needed. Topics come in to play when nodes need to communicate with each other. Topics are built upon a publish/subscribe system which allows nodes to publish data onto a certain topic. This data is then available for subscribing nodes to read and use for computing. This way of communication allows the nodes to talk with each

other without knowing who they are talking to. They just need to know what topic they need to publish and subscribe to in order to transmit and receive data from.

### **2.1.1.1. Master**

As it was mentioned before, ROS structure consists in the division of several sub-systems, therefore, it requires some controller to keep track of every node and topic within the system. This controller is called 'the ROS-master' and must always run in order for the ROS-system to operate [21].

The startup process that the master uses can be seen in figure 4. After booting up the master the first step is for the publishers to advertise the master that they are ready for publication on one or more topics. The master then creates the topic, but at this time no data is sent onto it since there is no subscriber. For the subscribers to be able to access data from topics a similar boot up process is executed. It sends a request to the master that it wants to subscribe to the topic and starts to listen to the topic.

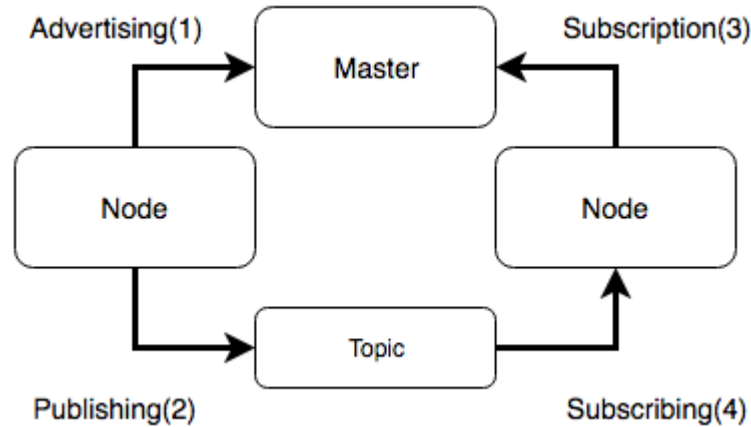


Figure 4. Publisher and subscriber initiation process

### **2.1.1.2.            *ROS commands***

The advantages of using ROS do not only include the way communication is handled but also its vast library of commands. These commands enable development, analyzation and simulation of algorithms to be executed [22]. Some examples of the commands are rosbag and rostopic.

The rosbag command allows the recording and playback of previously executed simulations [23]. Important to note that there are several more subcommands linked to rosbag that execute playback, filtering, etc.

## **2.1.2. APOLLO**

---

As previously mentioned Apollo Auto is a platform that enables development of AVs. Baidu provides a functioning system that can be directly implemented in a vehicle already provided, a Lincoln MKZ. For this aim a specific set of hardware is used. The importance of using the same vehicle and hardware is because of the way that Apollo transfers control signals to the vehicle is by CAN (Controller Area network). CAN is a network often used in vehicles to transfers control signals to everything from lights and wipers to motors and steering. The messages sent over the so called, CAN-bus vary a lot between different vehicles. This includes the structure, content and frequency of the messages. Another issue is the fact that for Apollo to run it needs feedback from the car. This includes speed, turning angle and acceleration, etc. The chance that the messages are constructed in the same way for separate vehicles is small and will therefore require customization.

### **2.1.2.1. Modules**

Modules can be defined as processes that perform the computation needed. In other words, they are the actual programs that operate with sensors, motors, and more. Modules are for Apollo what nodes are for ROS. Modules and nodes will be treated indistinctly from now on. Moreover, communication in Apollo is as essential, as it was in ROS. Therefore, modules are all related/connected at some point among them, so a communication needs to be established. This communication is similar to the one explained in ROS section based in topics.

Modules could be interpreted as a task division in a big process. The module division is one of the biggest strengths in ROS and in Apollo because of the flexibility they bring to the system. This means that a crash on any of the modules, wouldn't lead into an overall crash of the system, which makes the system safer and more reliable. In this case, for example, we have a module that defines the localization, another that recreates the map...

The end of this section is to explain some of the most important modules that are already made in Apollo and that were used during this bachelor thesis. Thus,

prediction, routing and planning will be explained in a general basis. However, as a main part of the thesis was to customize a map, the map module will be analyzed in more detail. Besides, the simulation module, Dreamview, will be explained in a separate chapter. It is important to know that there are other modules as Localization or Control, but they are out of the scope.

Perception module end is to recognize obstacles. Prediction and perception modules are very related, this is because prediction will receive the information of perception and will generate the most likely trajectory that the obstacle perceived could take.

The planning and the routing module will calculate a collision free route, so it can be executed and followed. For that purpose, it will need the information given by other modules such as map, perception... in the simulation a blue path is created for the Apollo vehicle to follow.

Another important module is the map module. To enable autonomous driving, a map is required by Apollo to plan a route that will take the vehicle safely from point A to point B. Apollo comes with a few pre-made maps on which a simulation can be run, but for a real-world scenario a map has to be made by the user. Apollo's maps are all based on the OpenDrive format. OpenDrive is an open file format for road-descriptions using xml (Extensible Markup Language). Apollo has made some modifications and as such the standard OpenDrive format will not work in Apollo. There is a definition sheet shortly describing the modified format which can be acquired from the Apollo team.

Map-data is split into three different versions. These are the base map (base\_map.xml), routing map (routing\_map.xml) and simulation map (sim\_map.xml). The routing map and the simulation map are simplified versions of the base map where only the most relevant data is kept. The base map is a complete map with all detailed coordinate-sets and map-elements, such as crosswalks, thoroughly defined. The simulation map is a compressed version of the base map, where coordinate-sets get shortened as much as possible for quicker execution during simulation. The routing map is a version of the base map with the possible routing-actions, such as lane switching. The routing module use this map to create a topological graph, describing the optimal route from point A to point B.

Apollo uses a collection of xml-parsers located in the map module to create the routing map and simulation map from the base map. These depend on the base map being an xml and as such there are scripts for translating a base map from either with a binary format or txt format to xml.

How Apollo is structured with which modules communicates with each other and via what topics can be seen in the table below.



<b>Module (Node)</b>	<b>Subscription</b>	<b>Publishing</b>
<b>Canbus</b>	/apollo/control	/apollo/canbus/chassis /apollo/canbus/chassis_detail
<b>Control</b>	/apollo/canbus/chassis /apollo/canbus/chassis_detail /apollo/planning /apollo/localization/msf_gnss /apollo/localization/msf_lidar /apollo/localization/msf_status /apollo/localization/pose	/apollo/control /apollo/cotrol/pad
<b>Localization</b>	None	/apollo/localization/msf_gnss /apollo/localization/msf_lidar /apollo/localization/msf_lstatus /apollo/localization/pose
<b>Map</b>	None	/apollo/relative_map /apollo/drive_event
<b>Monitor</b>	None	/apollo/monitor /apollo/monitor/static_info
<b>Perception</b>	None	/apollo/perception/obstacles /apollo/perception/traffic_light /apollo/sensor/gnss/best_pose /apollo/sensor/gnss/corrected_imu /apollo/sensor/gnss/gnss_status /apollo/sensor/gnss/imu /apollo/sensor/gnss/ins_stat /apollo/sensor/gnss/odometry /apollo/sensor/gnss/rtk_eph /apollo/sensor/gnss/rtk_obs /apollo/sensor/mobileye
<b>Planning</b>	/apollo/perception/traffic_light /apollo/prediction /apollo/routing_response /apollo/canbus/chassis /apollo/canbus/chassis_detail /apollo/localization/msf_gnss /apollo/localization/msf_lidar /apollo/localization/msf_status /apollo/localization/pose	/apollo/planning /apollo/routing_request
<b>Prediction</b>	/apollo/perception/obstacles /apollo/perception/traffic_light	/apollo/prediction
<b>Routing</b>	/apollo/routing_request /apollo/monitor	/apollo/routing_response

Table 1. Apollo publishing & subscribing structure

### **2.1.2.2.            *Communication within Apollo***

For standard ROS messages (.msg files) the structure always has to remain the same way [24]. In Apollo there are some changes on how communication is done. Apollo uses a message definition called protocol buffers, also known as *protobuf* in short, which allows for messages to be flexible. Flexibility means that the message has the ability to allow some structural changes. If one node does not send its complete message, in a normal setting this would cause the shutdown of the entire system in ROS, but thanks to the inherent flexibility of the protobuf messages it can still operate functionally given that the values missing are not critical for operation.

An example of how flexibility works in the case of an AV driving and receiving coordinates from a GPS of where it is located. Say that these coordinates are three dimensional (x, y, z). In many cases the car does not need to know at what height it is and therefore, if the z-coordinate is missing the car should operate as usual. This is possible thanks to protobuf messages.

Although protobuf enables flexibility and facilitates communication it creates some issues when integrated. One of these issues is that ROS requires published messages to be standard ROS messages and the output from modules in Apollo is protobuf. This problem is solved by using so called adapters. They will be explained in section 2.1.2.3.

While Apollo messages differs from the ROS standard structure of how modules communicate follows the same node/topic structure as ROS, only difference is that nodes are called modules. The structure of Apollo and how each module communicate with each other, i.e. which topics the modules publish and subscribe to, are presented in Table 1.

### **2.1.2.3.            *Adapters***

Due to the massive amount of data that an AV needs to handle because of the different sensors, cameras, lidars... The information needs to be recorded, stored and analyzed in an easy way. For this reason, Apollo decided to use protobuf messages. To establish a communication between ROS and Apollo Adapters were created. Adapters

can be viewed as translators of messages and are specialized for one specific topic. Hence there are several adapters per module so that each topic is translated correctly.

---

## **2.2.     *Hardware***

---

Since the Twizy does not have an internal computer with the capacity of running the algorithms needed for autonomous drive, this warrants a need for a separate computer.

The computer used for this project is a Novu-6801-GC with Nvidia GTX 1080 graphics card. The Novu is chosen for its durability, computing power and small form factor.

The Novu comes with most of the necessary components already installed, but some crucial components are not included at delivery. These are a CAN-card, SSD and graphics card. The CAN-card is used to convert messages into physical signals transported over the CAN-bus. The SSD is used for storage and the graphics card for heavy data processing. The hardware installation of these components was straight forward, but when it comes to software integration and installation there were some issues.

Initially, the installation process of Ubuntu ran without any problems. However, the Apollo kernel (linux-4.4.32-apollo-1.0.0) used in this project is optimized for Ubuntu 14.04 and the first installation of Ubuntu was done by using version 16.04. Because of this there were some problems with the graphics, greatly lowering the refresh rate. However, and installation of Ubuntu 14.04 fixed it.

When installing the CAN-card there were some issues with getting the CAN module to send messages. Since this problem affected several parts of the Twizy it was fixed in collaboration with another Twizy-group. The problem involved a function that converts messages from the control module to CAN-messages and vice-versa. When this function was called it only sent zeros over the CAN-bus and not the actual control messages that it was supposed to send.

The solution to this problem was to remove a large portion of the code which is located in a script called `vehicle_control.cc` between line 104 and 122. Those lines check

whether the messages that are supposed to be sent over the CAN-bus are valid or not. One consequence of removing the checks is that all messages gets sent over the CAN-bus, this includes messages that could become harmful for the vehicle and the surroundings. For example, if a test is run with simulated signals over the CAN-bus there is the possibility that the car drives blindly, which could lead to a crash. However, one advantage of this is that it enables tests of every node, except perception and location, and the vehicle itself.



Figure 5. IPC used

## Chapter 3 Simulation Module

In chapter 2 different modules were already introduced. This bachelor project has been focused specifically on the simulation module. Therefore, a new chapter has been opened with the purpose of explaining this module and the Dreamview interface.

Simulating in Apollo can either be done in the terminal using bag files, or in Apollos internal HMI, Human-Machine Interface, “Dreamview”, which can be seen in figure 6 [25]. There is a difference on what can be achieved in each and can thus warrant the use of both methods.

Dreamview is designed to visualize the data currently sent over the Apollo framework [25]. It provides the user with several options regarding choice of map and car (if several are located in the system) as well as the option to toggle what modules are currently running. This way of activating separate nodes makes possible the isolation of a node so it can simply be analyzed. These functionalities enable basic testing of control algorithms. The map used is a simplified version of the HD map, retaining only the core information for base functionality. As such is portrayed as a 2D map, with roads and obstacles visualized. Routes can be planned in Dreamview.

Bagfiles enables recording of real life scenarios, by driving the vehicle while recording how the system would react in each situation. By using the command "rosvbag-filter" one can filter out messages sent over topics related to a specific module and therefore allow for testing of new algorithms in specific modules [26]. This is useful in cases where there are several teams working on separate parts of software development that want to test only their own algorithms and not have to worry about bugs from another part of the system.

The main idea of this module is to provide the user a web application where the different outputs of the other modules are displayed. Apollos simulation uses a bash script called “dreamview\_sim\_control.sh” to enable simulation. The script uses Apollos HMI Dreamview, see figure below, for visualization. The activation of the simulation module is always required before simulating.

Using the terminal enables analysis of individual modules and topics in a more detailed way than by using Dreamview. This comes from the fact that the terminal outputs the actual message rather than a graphical view of the modules reacting. Because of this, each message can be deconstructed to locate possible errors.

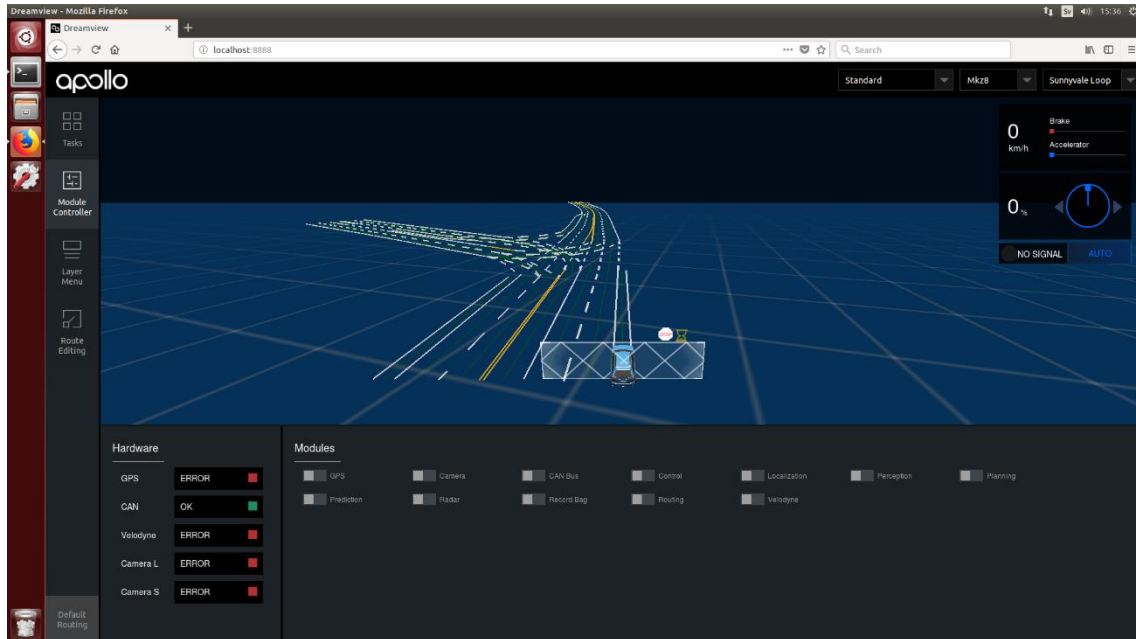


Figure 6. Dreamview interface

The table below contains the list of modules in which the Dreamview module is subscribe to and the list of modules where Dreamview publishes to.

Subscriptions	Publications
/apollo/localization/pose	None
/apollo/canbus/chassis	
/apollo/canbus/chassis_detail	
/apollo/planning	
/apollo/monitor	
/apollo/monitor/static_info	
/apollo/perception/obstacles	
/apollo/prediction	
/apollo/routing_response	

Table 2. Subscriptions and publications of Dreamview

### 3.1. Dreamview system notification scheme

---

Dreamview uses a visual notification system expressing the decision making that the car performs as it is driving both virtually and in reality [27]. For the planning module, decisions are depicted by *decision fences*, as illustrated in the left and middle picture in figure 7. The left fence is shown as the car is stopping for an obstacle on the road and the right fence denotes the decision of following the vehicle in front. The rightmost picture is shown as the car issues a signal to change lane.



Figure 7. Some of the symbols representing the decision making made by the system in the Dreamview visual notification scheme.

However, the car may stop for other reasons than simply detecting an obstacle in its path. A few of the symbols representing additional stopping reasons are shown in figure 8. As a stopping reason is issued, it will substitute the orange cone shown in the top right corner of the red stopping fence presented above. From left to right, the symbols denote stopping due to crosswalk in front, pedestrian crossing in front, destination arrival and emergency.

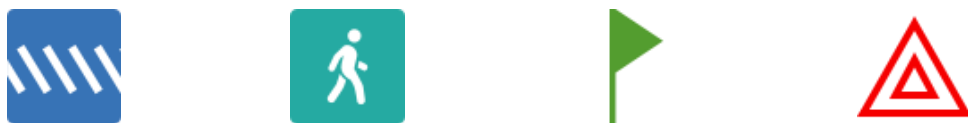


Figure 8. Symbols representing what causes the car to stop in the Dreamview visual notification scheme.

Perceived objects are assigned one out of four colors due to their type; vehicles are presented as green boxes, pedestrians as orange, bicycles as blue and unknown objects as purple. As the car is driving close to an object, an orange patch under the object appears, depicting an area that the car should avoid. If the perceived object is moving, the perception module will assign a white arrow to it, emanating from the geometrical center point and pointing in the direction that the object is heading. Likewise, the prediction module will assign a yellow arrow pointing in the direction of the predicted path.

There are several additional symbols that Dreamview can display in other situations. However, only the ones presented in this section were seen as relevant for this thesis. The tools to build and simulate scenarios are now acquired and, assuming the car behaves properly, it is clear which symbols should be displayed in which situations by virtue of this section.



## Chapter 4 Creating and testing a simulation environment

The main purpose of the project was to investigate ways of simulating and evaluating the system. In the beginning of this chapter, a simulation procedure using bagfiles is proposed. However, this method proves to be rather inconvenient and a more customizable simulation method is therefore required. The second method is based on the contents of section 4.2.1. and 4.2.2. A detailed comparison of the two methods is presented in section 4.2.3. The simulations are restricted to only run using the planning, prediction and routing modules. These are the only relevant modules to simulate since the other modules are interfaces to hardware, which is not included in the simulation.

More specifically, section 4.2.1. presents the process of creating a custom map. The map was used in conjunction with the second simulation method to create specific traffic scenarios. Three simulation scenarios are tested; a pedestrian crossing the road in front of the car, another car driving slowly in front of the car, another car driving slowly in front of the car and making the car perform a simple lane change. These are presented in sections 4.3.1, 4.3.2. and 4.3.3. respectively.

The aim of this chapter is to evaluate the quality of the simulation environment. The evaluation is primarily based on the content of section 3.1., which is a brief introduction to the Dreamview notification scheme, as well as on the scenario sections. At the end of each scenario, a scenario-specific evaluation is presented in section 4.4.

## 4.1. Simulation using bag files

As explained in chapter 3, it is possible to create a new bag file containing messages published on the topics of interest by filtering and existing bagfile. This can be used to simulate a module, by filtering out the input data and providing it to the module by playing the new bagfile. This procedure is explained by an example, using a bagfile included in Apollo 2.0. for demonstrative purposes.

The contents of the demo\_2.0.bag file can be inspected by running the command `rosbag info demo_2.0.bag` which yields the following output.

```
1 path:      demo_2.0.bag
2 version:   2.0
3 duration:  34.9s
4 start:     Dec 28 2017 22:37:38.00 (1514497058.00)
5 end:       Dec 28 2017 22:38:12.90 (1514497092.90)
6 size:      63.1 MB
7 messages: 46168
8 compression: none [80/80 chunks]
9 types:     pb_msgs/ADCTrajectory [97587fe9a5b2df2b61888d56c6fc697b]
10           pb_msgs/Chassis [d6a21658031a6a4615858d76f8b5178e]
11           pb_msgs/ContiRadar [cc92608f43dabc2a96119eaca0c19535]
12           pb_msgs/ControlCommand [67f7ff8a4c675dc97a8c7ce6d6289943]
13           pb_msgs/EpochObservation [6d088c32c2d00b8a760974f0c580dfb7]
14           pb_msgs/GnssBestPose [c4465e2257bee53c91729e53c69bf7c5]
15           pb_msgs/GnssEphemeris [aa53419c1014e11a3b4604e3c54daf7b]
16           pb_msgs/GnssStatus [6ab9bfa7e56e2724fd30280b731fef2]
17           pb_msgs/Gps [8fad5985ce947d3b6854fd093a59c429]
18           pb_msgs/Imu [bdef0ba51869607ed95736d41e80c1f5]
19           pb_msgs/InsStat [36306149a641468d85afa4cf44de7141]
20           pb_msgs/LocalizationEstimate [503c8e75900db180bc61534806a37cfb]
21           pb_msgs/LocalizationStatus [ea957896c739487ae64a8f4db8112e08]
22           pb_msgs/PerceptionObstacles [c6fd886a685be1dbbc6174bbc5a754de]
23           pb_msgs/PredictionObstacles [45bac0c01020cbf041fb9bd39f790e93]
24           pb_msgs/TrafficLightDetection [af38365a44e248f235d36014648275e9]
25           tf2_msgs/TFMessage [94810edda583a504dfda3829e70d7eec]
26 topics:   /apollo/canbus/chassis 3490 msgs : pb_msgs/Chassis
27           /apollo/control 3489 msgs : pb_msgs/ControlCommand
28           /apollo/localization/msf_gnss 35 msgs : pb_msgs/LocalizationEstimate
```

29	/apollo/localization/msf_lidar	175	msgs	: pb_msgs/LocalizationEstimate
30	/apollo/localization/msf_status	6535	msgs	: pb_msgs/LocalizationStatus
31	/apollo/localization/pose	6277	msgs	: pb_msgs/LocalizationEstimate
32	/apollo/perception/obstacles	348	msgs	: pb_msgs/PerceptionObstacles
33	/apollo/perception/traffic_light	105	msgs	: pb_msgs/TrafficLightDetection
34	/apollo/planning	349	msgs	: pb_msgs/ADCTrajectory
35	/apollo/prediction	348	msgs	: pb_msgs/PredictionObstacles
36	/apollo/sensor/conti_radar	465	msgs	: pb_msgs/ContiRadar
37	/apollo/sensor/gnss/best_pose	35	msgs	: pb_msgs/GnssBestPose
38	/apollo/sensor/gnss/corrected_imu	3490	msgs	: pb_msgs/Imu
39	/apollo/sensor/gnss/gnss_status	35	msgs	: pb_msgs/GnssStatus
40	/apollo/sensor/gnss/imu	6934	msgs	: pb_msgs/Imu
41	/apollo/sensor/gnss/ins_stat	35	msgs	: pb_msgs/InsStat
42	/apollo/sensor/gnss/odometry	3490	msgs	: pb_msgs/Gps
43	/apollo/sensor/gnss/rtk_eph	66	msgs	: pb_msgs/GnssEphemeris
44	/apollo/sensor/gnss/rtk_obs	69	msgs	: pb_msgs/EpochObservation
45	/tf	10398	msgs	: tf2_msgs/TFMessage

**Table 3. A printout of information about the demo\_2.0.bag file.**

The most essential parts of the information are the messages listed on line 9 to 25 and the topics on which they are published on, listed on lines 26 to 45. By consulting the table 1, the published topics in the bagfile can be compared to the subscription topics of the modules. The conclusion is that simulating every module using only the content from this bagfile is not possible. In fact, the only modules that could be provided with complete input data are planning, routing and prediction modules.

Suppose that the prediction module needs to be tested. Thus, a bagfile containing messages published on the */apollo/perception/obstacles* and the */apollo/localization/pose* topics needs to be created, according to the table 1. These topics are listed on line 31 and 32 in the information printout. Filtering is performed by running the following command.

```
$ rosbag filter demo_2.0.bag new_bag.bag
    'topic == "/apollo/localization/pose" or
    topic == "/apollo/perception/obstacles"'
```

Activating the prediction module by using e.g. the Dreamview module controller and playing the new bagfile by running the command `$ rosbag play new_bag.bag` will start the simulation. Echoing the */apollo/prediction* topic yields the output in the form of prediction obstacle messages as presented below.

1	header {	42	timestamp: 1514497088.83
2	timestamp_sec: 1521789098.58	43	predicted_period: 5.0
3	module_name: "prediction"	44	trajectory {
4	sequence_num: 447	45	probability: 1.0
5	}	46	trajectory_point {
6	prediction_obstacle {	47	path_point {
7	perception_obstacle {	48	x: 587590.958171
8	id: 1816 position {	49	y: 4140891.16965
9	x: 587590.958171	50	z: 0.0
10	y: 4140891.16965	51	theta: 1.36090444968
11	z: -30.522346882	52	}
12	}	53	v: 15.7597077
13	theta: 1.35948664408	54	a: -0.0368953225593
14	velocity {	55	relative_time: 0.0
15	x: 3.3099228761	56	}
16	y: 15.40820552	57	.
17	z: 0.0	58	.(several trajectory points)
18	}	59	.
19	length: 1.80951833725	60	trajectory_point {
20	width: 2.05610489845	61	path_point {
21	height: 0.663786709309	62	x: 587601.815494
22	polygon_point {	63	y: 4140967.35914
23	x: 587589.763831	64	z: 0.0
24	y: 4140890.56002	65	theta: 1.49771702947
25	z: -30.493086751	66	}
26	}	67	v: 15.7289489231
27	.	68	a: 0.0243981598557
28	.(several polygon points)	69	relative_time: 4.9
29	.	70	}
30	polygon_point {	71	}
31	x: 587591.76231	72	}
32	y: 4140890.0733	73	perception_error_code: OK
33	z: -30.531645177	74	start_timestamp: 1521789098.58
34	}	75	end_timestamp: 1521789098.58
35	tracking_time: 5.30291795731	76	
36	type: VEHICLE	77	
37	timestamp: 1514497088.83	78	
38	confidence: 0.925041079521	79	
39	confidence_type: CONFIDENCE_CNN	80	
40	}	81	
41		82	

**Table 4.** An output message from the prediction module during simulation using bagfiles.

Essentially, the prediction obstacle message wraps the perception obstacle, which contains information about the position, velocity, type, ID and geometrical shape of the object. The shape is determined by the polygon points, starting at line 23. The prediction module then adds additional information, starting at line 42, which is the predicted trajectory of the perceived object. The trajectory is represented by several trajectory points. In this particular message, the car is keeping track of a vehicle with ID 1816.

Simulating the planning or routing modules can be done in an analogous manner. Multiple modules can be simulated together as well by containing the input to each module in the same bagfile and playing it. While this method of simulation works, there are several substantial limitations to it. It is simply not possible to modify the location, velocity, acceleration, type, geometrical shape or path of any object in the

bagfile. A more convenient method is based on a custom map and obstacle placement function in the following two sections.

## 4.2. Creating the environment

The main purpose of the project is to create a simulation environment, a customize map and obstacles placed on that map. In order to achieve this goal some coding was done. The scope was to recreate the road in Chalmers but due to construction work on one of the roads constituting the car route which was initially decided upon, the teams were forced to revert to driving on a simple straight on campus. This chapter 4.2. will explained in detail how the custom map and the obstacles were created and integrated in Dreamview.

### 4.2.1. Creating a road

By investigating the code of one of the already existing base\_map.xml files in the system, another Twizy project group managed to create a basic map consisting of a single lane straight which the car was able to route and drive on in simulations. The group also managed to obtain a specification sheet on the OpenDrive format used in Apollo. Using these tools, the coordinate system of the map was changed, and an additional lane and a crosswalk were included. This resulted in a road structure as illustrated in figure 9.

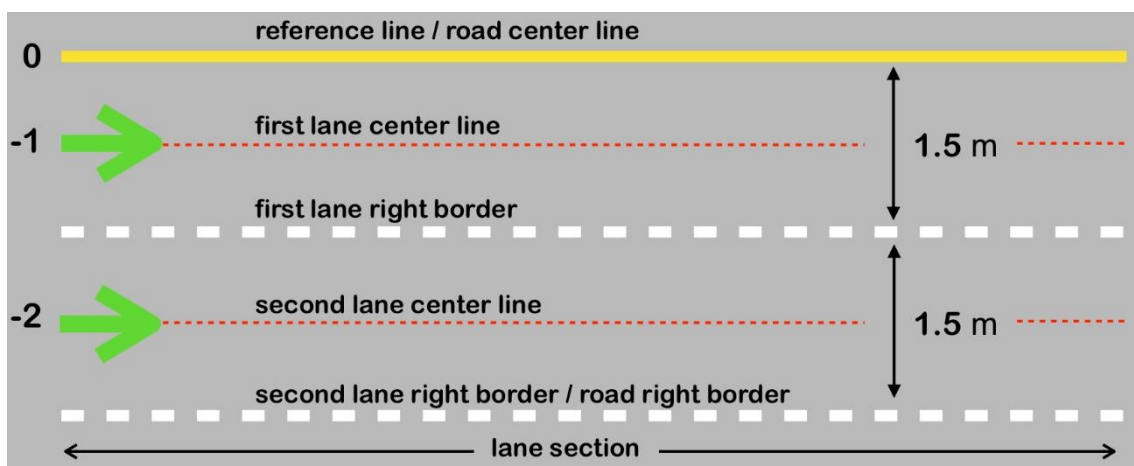


Figure 9. The structure of the virtual wo-lane road used in the simulations.

The final code for the `base_map.xml` file is listed in the Appendix B.1. The code begins with some basic xml file declarations and a header containing meta-data. On lines 15 through 17, there is special geographical meta-data. Starting from the right of line 16, it is declared that the unit of measurement used in the subsequent code is meters. The coordinate system is chosen according to the UTM standard, declared by the assignment to `proj`, which makes the map compatible with Dreamview.

```
15     <geoReference>
16     <![CDATA[+proj=utm +zone=31 +ellps=WGS84 +datum=WGS84 +units=m +no_defs]]> 17
17     </geoReference>
```

Line 19 declares the road that holds the two lanes and assigns the road identifier `r1` to it. Lines 22 to 29 specify a crosswalk of width 2m located at a distance of 40 m from the starting end of the straight.

```
22     <object id="1" type="crosswalk">
23     <outline>
24     <cornerGlobal x="40" y="0" z="0" />
25     <cornerGlobal x="40" y="-6" z="0" />
26     <cornerGlobal x="42" y="-6" z="0" />
27     <cornerGlobal x="42" y="0" z="0" />
28     </outline>
29     </object>
```

On lines 32 to 34, the `lanes` tag holds the `laneSection` which holds the road *boundaries*. On lines 36 to 53, the leftmost and rightmost boundaries of the road are declared. By OpenDrive convention, the center line of a road is a lane identifier 0 and no width, as declared on line 56. This line is also referred to as the *reference line*.

```
56     <lane id="0" uid="r1_1_0" type="none">
```

Since the road contains two lanes with identical driving direction, no overtaking by crossing the road center line is allowed and the line is set as a solid yellow on line 64.

```
64     <borderType sOffset="0" type="solid" color="yellow"/>
```

The innermost lane is declared at line 70 and assigned the lane identifier -1 by convention.

```
70     <lane id="-1" uid="r1_1_-1" type="driving" direction="forward" turnType="noTurn">
```

Lines 71 to 78 declare the lane center line. The left border of the lane is hard limited by the road reference line, but the right border may reside any distance from the

reference line as long as it is not further away than the rightmost road border. On lines 79 to 87, the first lane right border is set as to split the road in two parts of equal width 1.5 m.

```
79 <border> <!-- Lane right side border -->
80 <geometry sOffset="0" x="0" y="-3" z="0" length="1000">
81 <pointSet>
82 <point x="0" y="-3" z="0"/>
83 <point x="1000" y="-3" z="0"/>
84 </pointSet>
85 </geometry>
86 <borderType sOffset="0" type="broken" color="white"/>
87 </border>
```

The declaration of the second lane on lines 104 to 138 follows the same pattern as the first. The second lane has assigned the lane identifier 2 by convention.

As the base map is complete, the routing map is generated by running the command

```
./scripts/generate_routing_topo_graph.sh --map_dir ${dir_name}
```

where the parameter `dir_name` is the path to the directory holding the `base_map.xml` file. The command yields two new files, `routing_map.txt` and `routing_map.bin`. Similarly, the map used for Dreamview visualization is generated by the command

```
$ bazel-bin/modules/map/tools/sim_map_generator --map_dir=${dir_name} --
output_dir=${dir_name}
```

which yields the files `sim_map.txt` and `sim_map.bin`. To ensure that the maps have been generated correctly, the terminal output should be investigated. Using the base map presented in this section yields the output listed in table 5. Lines 1 to 14 and 15 to 33 represent the results from generating the routing map and the sim map respectively. As evident from lines 3 and 4 as well as 15 and 16, the two maps contain one road, two lanes and one crosswalk. Lines 18 to 31 ensures the success of the down sampling needed to create the sim map.

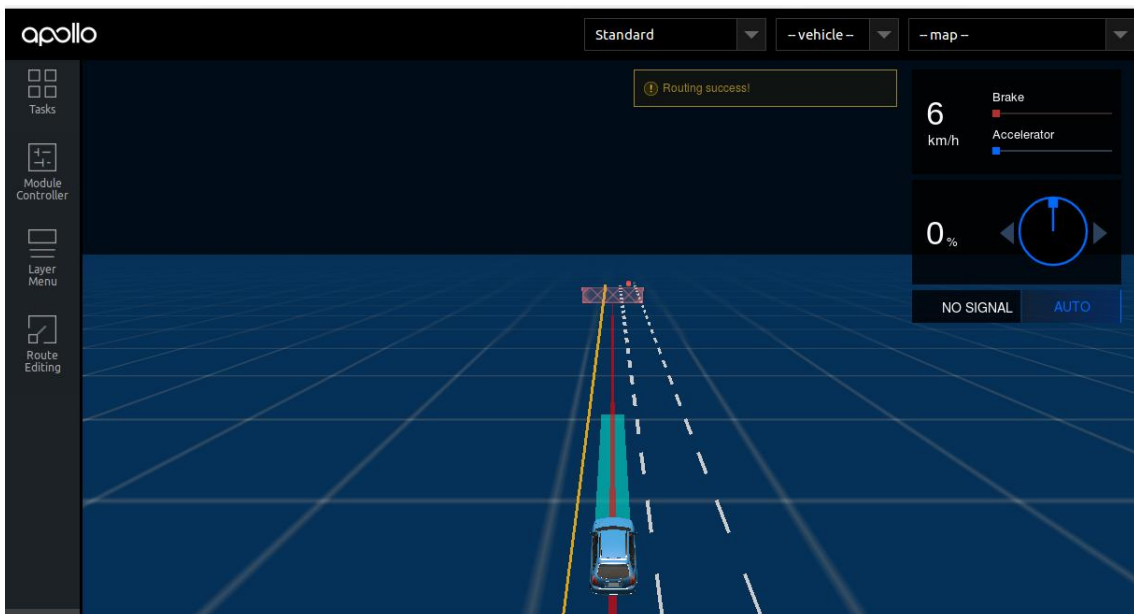
```
1 I0508 12:32:49.797320      119 topo_creator.cc:36] Conf file:
2 modules/routing/conf/routing_config.pb.txt is loaded.
3 I0508 12:32:49.801218      119 proto_organizer.cc:373] hdmap statistics: roads-1,lanes-2,
4 crosswalks-1,clear areas-0,speed bumps-0,signals-0,stop signs-0,yield signs-0,
5 junctions-0,overlaps-3
6 I0508 12:32:49.801396      119 graph_creator.cc:72] Number of lanes: 2
7 I0508 12:32:49.801429      119 graph_creator.cc:98] Current lane id: r1_1_-2
8 I0508 12:32:49.801462      119 graph_creator.cc:98] Current lane id: r1_1_-1
9 I0508 12:32:49.803390      119 graph_creator.cc:146] Txt file is dumped successfully. Path:
10 modules/map/data/Simtest/routing_map.txt
11 I0508 12:32:49.803647      119 graph_creator.cc:151] Bin file is dumped successfully. Path:
12 modules/map/data/Simtest/routing_map.bin
```



13	I0508 12:32:49.803679	119 topo_creator.cc:43] Create routing topo successfully from
14		modules/map/data/Simtest/base_map.xml to modules/map/data/Simtest/routing_map.bin
15	I0508 12:32:49.875528	121 proto_organizer.cc:373] hdmmap statistics: roads-1,lanes-2,
16		crosswalks-1,clear,areas-0,speed bumps-0,signals-0,stop signs-0,yield signs-0,junctions-0,
17		overlaps-3
18	I0508 12:32:49.875799	121 sim_map_generator.cc:84] Downsampling lane r1_1_-2
19	I0508 12:32:49.875811	121 sim_map_generator.cc:77] Lane curve downsampled from 2 points
20		to 2 points.
21	I0508 12:32:49.875821	121 sim_map_generator.cc:77] Lane curve downsampled from 2 points
22		to 2 points.
23	I0508 12:32:49.875828	121 sim_map_generator.cc:77] Lane curve downsampled from 2 points
24		to 2 points.
25	I0508 12:32:49.875849	121 sim_map_generator.cc:84] Downsampling lane r1_1_-1
26	I0508 12:32:49.875854	121 sim_map_generator.cc:77] Lane curve downsampled from 2 points to
27		2 points.
28	I0508 12:32:49.875875	121 sim_map_generator.cc:77] Lane curve downsampled from 2 points to
29		2 points.
30	I0508 12:32:49.875880	121 sim_map_generator.cc:77] Lane curve downsampled from 2 points to
31		2 points.
32	I0508 12:32:49.877517	121 sim_map_generator.cc:121] sim_map generated at:
33		modules/map/data/Simtest

**Table 5. The output after generating the routing and sim maps.**

As it is possible to create a map consisting of any road structure of choice, it would be natural to include obstacles in the environment. While this could be done by using bagfiles to publish perceived objects on the /Apollo/perception/obstacle topic, the second method is far more flexible in conjunction with the custom map.



**Figure 10. Straight Lane in Dreamview**

## 4.2.2. *Obstacle implementation*

---

The second method offers the ability to change several features of a perceived object, as opposed to the bagfile method. Parameters such as the position, velocity, acceleration, path, shape, type and ID may all be chosen, greatly simplifying the creation of traffic scenarios.

A new script called `sim_test.cc` containing a function called `updateSimWorld` was created, enabling the desired creation and placement of box-shaped obstacles in the simulation environment. The code for the script and function is listed in appendix C.1 and C.2. Lines 1 to 15 declare the files that needs to be included. For example, the corresponding header file called `sim_test.h` is included on line 1. The current files are located in the Dreamview module directory, requiring the innermost namespace Dreamview on line 18.

The declaration of the new main function is made at line 32. It is followed by the declaration of a `PerceptionObstacles` object which is a form of container holding multiple `PerceptionObstacle` objects. An additional function named `addPerceptionObstacle` is called on lines 36 and 55. The call on line 36 relate to the scenario presented in section 3.6 and the call on 55 to the scenario in section 4.3.2.

```
36 addPerceptionObstacle(&obstacles, ::apollo::perception::PerceptionObstacle::PEDESTRIAN, 1, 41, y, 0, 0, 1, 0, 1, 1, 1, 1);
```

In calling the function, the first parameter is a `PerceptionObstacles` object, the second is the type of object that is perceived, and the third parameter is an object identifier. On line 36, the type of object is a pedestrian with the identifier 1. The three following parameters are the x-, y- and z-coordinates of the object's geometrical center point in a Cartesian coordinate system. The pedestrian is moving and therefore the y-coordinate is updated according to the parameter y. The succeeding three parameters describe the velocity in the x-, y- and z-direction in meters per second. If the object is stationary, the `addPerceptionObstacle` function with 12 parameters on line 65 is indirectly called with zero velocity in each direction. The final three parameters describe the width, length and height of the box that represents the object, measured in meters.

The `addPerceptionObstacle` function sets certain parameters of the passed `PerceptionObstacles` object, as described by the input parameters. This object is then published on the `/apollo/perception/obstacles` topic on line 71, ending the main function body.

The main function publishes only once per call and must be called upon frequently enough in order to mimic an object perceived in real time. After some investigation in the `Dreamview` module directory, a script called `sim_control.cc` was found, containing a convenient function that runs as often as the car's position is updated. Calling the self-made `updateSimWorld` function inside this function provides the publishing rate of the `PerceptionObstacle` messages that is needed. The `sim_control.cc` scripts are listed in appendix C.3.

Originally however, the `Dreamview` module is not permitted to publish any messages onto the `/apollo/perception/obstacles` topic. To solve this, a file called `adapter.cong` located in the `Dreamview` module directory was modified by changing the communication mode from `RECEIVE_ONLY` to `DUPLEX`. Since changing code in the system generally requires rebuilds, the new `sim_test.cc` and `sim_test.h` files were added to the local `BUILD` file.

By rebuilding the system and initiating the simulation, the custom objects will be shown in `Dreamview` as if the car is actually perceiving them. However, the perception module is bypassed and not active in any way. While the perception module in fact can be simulated, pre-recorded sensor data is required in order to do so. Being limited to the scenario offered by the sensor data would contradict the purpose of the simulation method; to be able to easily create and test new traffic scenarios.

The two simulation methods have been presented and their differences have been mentioned briefly. However, a more detailed comparison is presented in the following section. The comparison is explained by an example.

### ***4.2.3. Simulation methods advantages and disadvantages***

---

By the use of the `sim_test.cc` script, the ability to customize the position, velocity, acceleration, type and geometrical shape of the object was acquired. This is the first key aspect to why the second simulation method is superior to the bagfile method. The second aspect involves the map. In fact, creating traffic scenarios by using tailored according to the obstacles placed in it instead of the opposite. This may be explained by a proposition on how to recreate the scenario of the crossing pedestrian, assuming that a custom map is used.

Initially, a bagfile such as the `demo_2.0.bag` file would need to be played and Dreamview accessed. The environment would then need to be searched until a pedestrian was found. The pedestrian must not be close to any other object, since objects cannot be removed from the bagfile and the scenario should include the pedestrian and the car only. Furthermore, the walking path of the pedestrian must be somewhat straight. Due to these restrictions, the likeliness of finding an adequate situation is already marginal.

Presume that a pedestrian has been found. Since the pedestrian should walk across the road, the road must be placed accordingly. If the road is a simple straight, placing the road will not require too much effort. However, placing a road with a more complex structure including e.g. junctions, curves and roundabouts could be extraordinarily time consuming and intricate. Assuming that this problem has been solved, the scenario may then be simulated.

Most likely however, the first iteration of the scenario will not be completely satisfying. For example, the map might need to be finely adjusted to suit the walking path of the pedestrian. Furthermore, the pedestrian will re-spawn only as frequent as the bagfile is re-played, possibly making it hard to synchronize the car to encounter the pedestrian at a specific moment. The rate of spawning cannot be changed, since the duration of the bagfile cannot be modified.

To conclude, the combined use of a custom map and the `sim_test.cc` script makes it relatively easy to simulate a multitude of traffic scenarios. Before presenting the traffic, scenarios chosen in this report, it is important to ensure that the quality evaluation

---

of the environment will be done in a correct way. As will be shown, the evaluation is based on inspection of module output as well as on the notification system used in Dreamview. The latter requires a general introduction, after which the scenarios will be explained in more detail.

---

## 4.3. Complete system simulation

---

The goal of this bachelor thesis is the simulation of some real-life scenarios, so money can be saved by seeing the behavior of the vehicle in the computer instead of developing some real prototypes and traffic situations which cost vast amounts of money.

This last stage of the simulation work was focused on developing those scenarios. There are four different scenarios that will be explained in detail during this section.

### 4.3.1. Scenario 1: Pedestrian crossing the road

---

This particular scenario was chosen due to its frequent occurrence in real world traffic. It is also interesting in the wake of the fatal accident in which a self-driving Uber car ran over a pedestrian crossing a road at night in Arizona, USA this year [28].

In the `sim_test.cc` file, a line instantiating a moving obstacle shaped like a cube with side 1m was added, which represents the crossing pedestrian. The pedestrian moves across the crosswalk at a fairly slow pace. The pedestrian as an object and the mechanics that makes it move are represented by the code on lines 46 to 51.

The scenario is illustrated in figure 11. Initially, a routing request has been sent and the car accelerates as the routing and planning modules have decided on the preferred trajectory. In picture 1 and 2, the car starts to decelerate as to be able to stop in time for the crossing pedestrian. Note that the color of the box representing the pedestrian is orange as expected, but that the white arrow is pointing perpendicularly to the pedestrian's actual heading. Note also that the stopping reason is due to *an obstacle in front* and neither as specific as to a *crosswalk in front* nor to a *pedestrian crossing in front*. In picture 3, the car is standing completely still. As the pedestrian reaches the center line of the second lane in picture 4, the car finds it safe enough to start moving again. As it drives past, Dreamview displays the orange avoidance zone below the pedestrian. By closer

inspection of the last picture, it is possible to observe the green flag in the top right corner of the red decision fence ahead, the *stopping due to destination arrival* symbol.

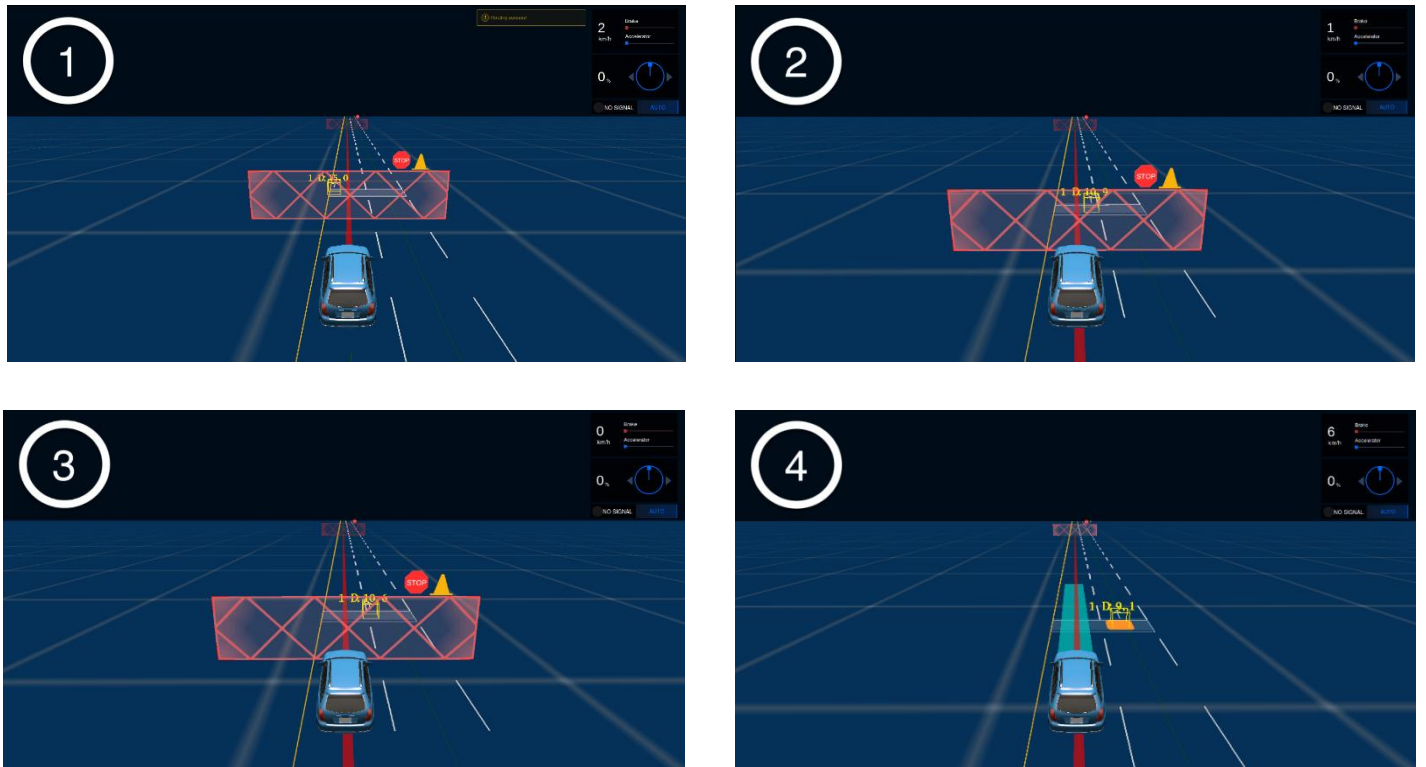


Figure 11. Four important moments in time from the scenario of the crossing pedestrian

The decisions of the car may also be inspected in text format by printing out the output of the planning module, as listed in table 6.

<pre> 1 decision { 2   main_decision { 3     stop { 4       reason_code: STOP_REASON_OBSTACLE 5       reason: "stop by 1" 6       stop_point { 7         x: 34.5 8         y: -1.500001 9       } 10      stop_heading: 9.43860757827e-15 11      change_lane_type: FORWARD 12    } 13  } 14  object_decision { 15    decision { 16      id: "1" 17      perception_id: 1 18      object_decision { 19        stop { 20          reason_code: STOP_REASON_OBSTACLE 21          distance_s: -6.0 22          stop_point { 23            x: 34.5 24            y: -1.500001 25          } 26          stop_heading: 9.43860757827e-15 } 27        } 28      } 29    } </pre>	<pre> 30 decision { 31   id: "DEST" 32   perception_id: -1109540454 33   object_decision { 34     stop { 35       reason_code: STOP_REASON_DESTINATION 36       distance_s: -0.5 37       stop_point { 38         x: 64.1356383595 39         y: -1.500001 40         z: 0.0 41       } 42       stop_heading: 6.60935984861e-16 43     } 44   } 45 } 46 } 47 vehicle_signal { 48   turn_signal: TURN_NONE 49 } 50 } 51 } 52 } 53 } 54 } 55 } 56 } 57 } 58 } </pre>
--	---

Table 6. Output message from the planning module when stopping dor the crossing pedestrian

The message contains information about several decisions. Essentially, each decision consists of a stopping reason, a stopping point and the current distance to the stopping point. In this case, the first stop decision is marked by *STOP\_REASON\_OBSTACLE* on line 20 and the second by *STOP\_REASON\_DESTINATION* on line 35. Thus, there is a proper correlation between the output from the planning module and the decision fences as seen in the pictures in figure 11.

The scenario was then repeated with a modified version of the road, the maximum speed set to 50 km/h as opposed to the 10 km/h used previously. As the car carries a higher speed, it needs to brake harder when coming up on the crossing pedestrian. The car managed to stop in time in each run, but the sequence of notifications and signals did not differ from the runs with the lower speeds. Note that the car did not reach 50 km/h but rather 20 km/h due to the short run down to the crosswalk.



## Evaluation:

As the car encounters the crossing pedestrian, it is reasonable to assume that the symbol of *stopping due to a pedestrian crossing in front* should appear. Instead, the symbol for *stopping due to an obstacle in front* is displayed. While stopping for an obstacle is not untrue, it is not as specific as might be desired. Furthermore, the object is in fact recognized as a pedestrian by the perception and prediction modules, which makes the absence of the pedestrian symbol contradictory. A likely cause of the problem is potentially missing information about the pedestrian as it is declared and published in the `sim_test.cc` file. As a matter of fact, not every parameter that is available in the PerceptionObstacle protobuf message was assigned a value. Thus, the information that the planning module receives about the pedestrian might be insufficient to determine the true stopping reason.

Likewise, it is not unreasonable to assume that the symbol for *stopping due to a crosswalk in front* should appear. An initial suggestion would be that some error in the `base_map.xml` file is causing the absence of the symbol. Unarguably, the crosswalk does not visually appear as a crosswalk, but rather as a slightly opaque rectangle. While this may cause suspicion, the crosswalks in each of the original Apollo 2.0. maps have identical appearances. Furthermore, the results of generating the routing and sim maps, as was presented in table 5, shows that the road indeed contains a crosswalk. Thus, from a structural point of view the `base_map.xml` is likely to be successfully implementing the crosswalk. From a simulation point of view however, further comparisons must be made.

A quick test of the Sunnyvale Loop, one of the maps in the original collection, produced no symbols when driving past crosswalks. However, the test did not include any crossing pedestrians. Including pedestrians by playing the `demo_2.0.bag` file or using the `sim_test.cc` script proved to be fruitless. No crossing pedestrian could be found in the environment when playing the bagfile and the `sim_test.cc` script could not be used due to the messages potentially missing some information. Thus, it is not possible to make a decent comparison of the Sunnyvale simulation run and the first scenario until a solution to either of these two problems is found.

As stated, there are probably no errors with the crosswalk in the `base_map.xml` file. Furthermore, it is not completely clear whether the crosswalk symbol should appear in the scenario at all. A quick comparison of one of the original Apollo 2.0. maps with the

custom map used in the scenarios was aimed at providing an answer to the matter. However, due to some complications a reasonable comparison could not be made. Thus, to be able to conclude anything about the crosswalk symbol, the problem of the incomplete messages should be solved.

The cause of the white arrow pointing continuously in the wrong direction is likely an error in the `sim_test.cc` script. It might also be the results of missing information about the pedestrian object. The orange patch is visible as the car gets close to the pedestrian as expected.

Increasing the speed limit on the road was of interest to investigate whether the emergency symbol presented in figure 8 would appear. Hypothetically, as the speed limit increases, the car will carry more speed and eventually need to brake hard enough for the symbol to be presented. As previously mentioned, the distance between the start of the straight and the crosswalk was not long enough for the car to reach above 20 km/h, regardless of the speed limit. Furthermore, it is unclear which type of scenarios are classified as emergencies. Thus, there is no way of properly evaluating the environment in this situation. To be able to conclude anything, the crosswalk would need to be moved further away from the start of the straight.

To summarize, the environment is solid enough in order to make the car drive safely on the road. However, the symbol for *a crossing pedestrian in front* does not appear as the car stops for the pedestrian. While the symbol for a crosswalk in front does not appear either, it is unclear whether it is supposed to be displayed at all. Most likely, the cause of the pedestrian symbol error is missing information about the perceived object. An orange patch is visible below the pedestrian, but the white arrow is pointing in the wrong direction. It could not be concluded if the speed of the car has any implications on whether the situation is classified as an emergency.

### 4.3.2. *Scenario 2: Following another car*

---

One safe assumption regarding AV's would be that not every car in the world would instantly become autonomous once AV's hit the road. As such, a core functionality required of an AV would be to be able to handle situations involving other vehicles, autonomous or not. To demonstrate how Apollo handles this, a scenario involving another vehicle is shown here. As this scenario includes another car the following naming convention is used, "Apollo car" and "obstacle car".

In this scenario the obstacle car is spawned 20 meters ahead of the Apollo car. It moves at a slow pace of around 3-4 km/h and the Apollo car is initially following it, as illustrated in picture 1 in figure 12. Note that Dreamview displays the *red stopping fence* instead of the *green follow fence*. In picture 2, the obstacle car starts drifting into the other lane and once merged, it continues along the second lane for a short while. In picture 3, the obstacle car is cleared, and the Apollo car removes the need for slower speed. Note that the white arrow is continuously pointing in the same direction as in the last scenario. However, the orange patch under the obstacle car is properly displayed. In the last picture, the Apollo car has accelerated and is approaching its destination.

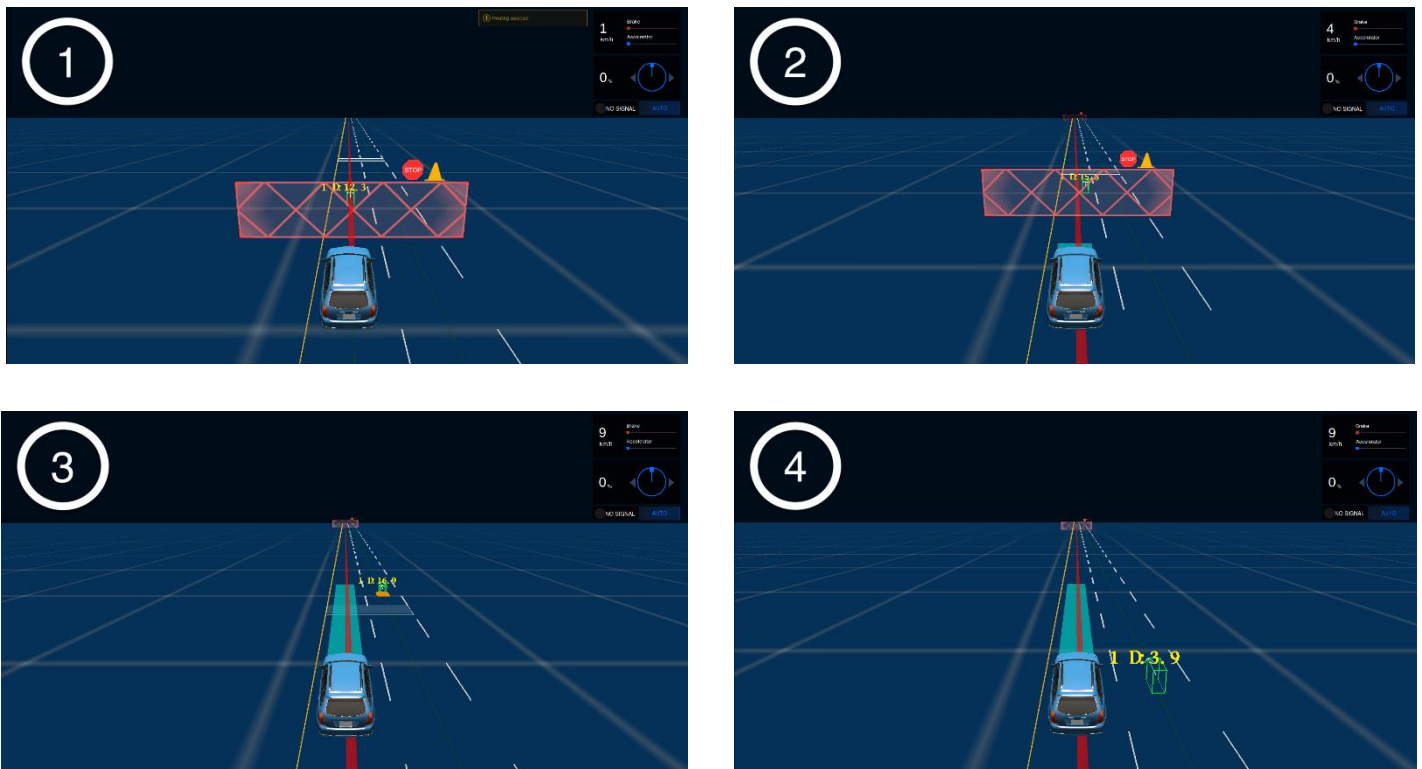


Figure 12. Four important moments in time from the scenario chasing another car

## Evaluation:

It is reasonable to assume that the *green follow fence* depicted in figure 7 should appear instead of the *red stopping fence* in the same figure. What is interesting is that Apollo car actually manages to follow the obstacle car regardless. However, it does so in a seemingly unconventional way by moving the stop fence in front of it as it drives. A likely cause of the fence error may once again be due to missing information about the obstacle car when it is published in the `sim_test.cc` file, somehow making it impossible to be followed properly.

Once again, the white arrow is pointing in the wrong direction. In fact, it is pointing identically to the white arrow in the previous scenario. Thus, it is also likely to be the results of the same error.

### 4.3.3. Scenario 3: Change of lane

---

Switching between lanes is a common action in everyday driving. The purpose of the scenario is to determine whether the vehicle notice about the change of lane. For this aim, the output of the routing module is analyzed.

Initially, the car is requested to drive from the left lane to the right by sending a routing request. Echoing the `/apollo/routing_response` topic yields a message of which a selected part is listed in table 7. Essentially, the route is based on two *passages* on line 8 and changing to the right lane. Line 10 declares the starting point as located in the left lane. The change of lane is indicated on line 14. The car that reaches the next passage, consisting of the right lane as declared on line 18. From here it will proceed to its destination without changing lane again, as declared on line 23.

```
1  header {
2    timestamp_sec: 1525439337.15
3    module_name: "routing"
4    sequence_num: 5
5  }
6  road {
7    id: "r1"
8    passage {
9      segment {
10     id: "r1_1_-1"
11     start_s: 120.246563791
12     end_s: 167.255089008
13   }
14   change_lane_type: RIGHT
15 }
16 passage {
17   segment {
18     id: "r1_1_-2"
19     start_s: 120.246563791
20     end_s: 167.255089008
21   }
22   can_exit: true
23   change_lane_type: FORWARD
24 }
25 }
```

Table 7. Parts of a routing response message from the routing module after sending a routing request to switch lanes.

After several runs of the simulation, it was concluded that the change of lane is not fulfilled every time. In the erroneous runs, the route end point is not saved correctly after the routing request has been sent. Thus, the car cannot interpret the path established by the user which causes it to drive in the lane it started in without ever stopping.

## Evaluation:

The green symbol representing the decision to change lane as depicted in figure 7 is not shown as the car switches lanes. However, the symbol does not appear when switching lanes in the any of the original Apollo 2.0 maps either. Thus, it is likely that the symbol appears in related circumstances, such as needing to switch lane when overtaking a slow car in front.

The issue of the car not always being able to switch lanes might be the implication of some data not being reset in the planning or routing modules while re-routing. To conclude, the simulation environment most likely enables the car to switch lanes but is interrupted from time to time by some unknown issue. However, the issue is likely not the results of an error in the simulation environment.

---

## ***4.4. Conclusions and general discussion***

---

To conclude, the simulation environment enables the car to drive safely on the road. However, the information that is displayed in Dreamview is not as specific as is expected. It is deemed likely that some of the missing information about the published obstacles is causing the planning module not to fully comprehend the situation. This is in turn causing the absence of the correct symbols. Nevertheless, the `base_map.xml` file seems to be properly implemented. The cause of the intermittent lane switching issue is uncertain but suggested to be a type of reset issue related to the module.

One way of further investigating the missing information problem is simply to compare the parameters in the published messages with the PerceptionObstacle messages from other sources. The `demo_2.0.bag` file, whose content was listed in table 3, is one such promising source. Another approach is to look at the PerceptionObstacle protobuf message definition, which is found in the perception module directory. The definition contains declarations of all parameters that are allowed in the message.

The reason a straight was created for the simulations was due to the simplicity of implementing it and that now of the chosen scenarios demanded anything more complex. By all means, scenarios such as making the car drive to a junction and turn onto a road with dense traffic or driving the car in a roundabout together with other vehicles could be created. While simulating the car in multiple scenarios would certainly be interesting to evaluate the car's behavior, it is important to note that this was not the aim of testing the simulation environment.





## Chapter 7 ENVIRONMENTAL IMPACT

There is a main concern about the rise in CO<sub>2</sub> emissions. One of the primary sources where they come from is the burnt of fossil fuels. A lot of targets had been accomplished through the years, the most recent one is the Paris Agreement. They had set an increase of the global temperature in 2 degrees Celsius [32]. One of the main sources of CO<sub>2</sub> emissions is the transportation sector. According to the Europe commission, cars are responsible for around 12% of total EU emissions of carbon dioxide. However, emerging countries demand in this sector is increasing. [33]. The image 20000 remarks the share of road transport estimation in a worldwide scale [34].

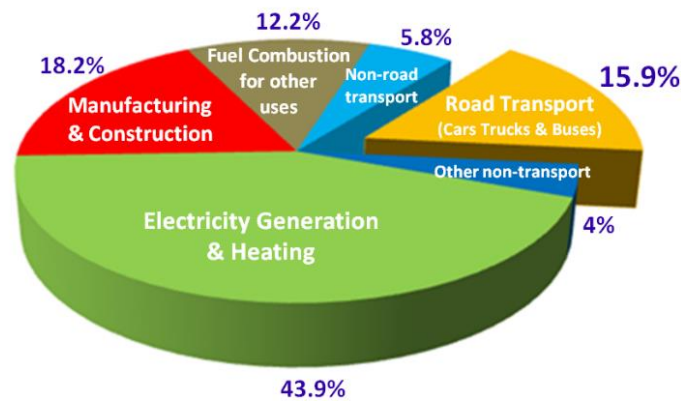


Illustration 7. Man-made CO<sub>2</sub> emissions

Due to all the concern that was mentioned and explained above, the automation industry has taken steps, vehicle manufacturers, for example, have already reduced pollutant emissions. This achievement is based on fuel combustion improvements.

Coming into autonomous driving a lot of fuel could be saved. There are a lot of possible explanations for the reduction of the emissions. Firstly, a higher efficiency with a faster flow is expected, besides, traffic congestions could be reduced. This would be achieved with the vehicle-to-vehicle communication. Systems would be able to identify busy routes, so they can avoid them. DHL explains in their article that with fuel efficiency achieved by optimized driving and by convoying, owners of driverless vehicles can reduce their carbon footprint and motoring costs by approximately 15% [35]

Another way of using efficiently the existing energy sources would be the charging moment. The idea here is that the vehicle would strategically be charged at times of low demand. This is normally when renewable energy is produced. Therefore, no additional power capacity may be needed. This could lead into an increase of renewable projects [36].

Vehicles could become lighter, they would not need any extra protection for crash avoidance. Therefore, the materials used for their fabrication would experience a significant decrease. Moreover, they will be equipped with a electric propulsion system that could reduce the CO<sub>2</sub> emissions up to a 95% [37].

We could conclude, hence, that the implementation of the AVs would mean a great advanced in the CO<sub>2</sub> emissions reduction. This would support even more the research in the autonomy sector. It should be said that all this are pure factual conjectures because of the lack of AVs driving the streets.

## Chapter 9 CONCLUSION

Since the task of creating an AV is large and complex there existed the need for a platform to use when creating the system for the autonomous Twizy. As a consequence of this Apollo Auto was chosen by the project manager. The idea was that Apollo included a lot of the necessary code already complete to operate the Twizy. This choice came with pros and cons. Apollo being a relatively new project with updates coming daily, a difficult barrier to overcome. To cope with the lack of documentation there was a lot of trial and error through the project. This was time consuming and required changes to the original project plan.

Initially, the project plan allocated a great deal of time for integrating new code into the system. However, during the course of the project it became evident that few of the Twizy project groups would provide new code that needed to be integrated. Most of the new code was implemented in ways that kept the same inputs and outputs that Apollo originally comes with. The sole group that eventually provided new code, which required integration, found it easier to perform the integration themselves with only minor intervention from us. The implications were several rearrangements of the project plan and an increased focus on the simulation part of the project.

To be able to drive the route on campus the car needs a map. Since it was unclear which group was responsible for creating this map, we offered to do it in collaboration with another group. As the project proceeded the route had to be changed due to construction work. A simple, straight street on campus was chosen as the new route. Since the street is a straight, the process of creating the map was identical to the process of creating the map in chapter 4. However, the points constituting the road are GPS coordinates collected by the other project group. The map is found in appendix B.2.

The final part of the projects goal was to create a model of the Twizy that could be implemented into the simulation environment. This was to make the car in the simulation as representative of the Twizy as possible. After a while however, another project group was assigned to create the model which left only integration of the model into Apollo.

The main part of the project was to create a simulation environment consisting of a map, a virtual model of the Twizy and static and dynamic obstacles. In chapter 4, a

map of a long straight was created. The chapter also presented two methods to successfully include both stationary and moving objects in the environment. Based on simulations of three traffic scenarios, the quality of the environment was evaluated. While the car was able to drive safely on the road in each scenario, some of the expected symbols did not appear. The cause of the error is most likely some missing code in the `sim_test.cc` script. However, the `base_map.xml` file seems to be properly implemented. Apart from not creating a virtual model of the Twizy, we therefore consider the aims regarding the simulation environment to be achieved.

## Chapter 10 FUTURE DEVELOPMENTS

As stated earlier the documentation of Apollo is scarce and badly structured. Hence focus was placed on creating a guide that can be used in future projects that use the Apollo platform. This work resulted in the how-to document located in appendix E.

Possible future projects are many as the simulation has some flaws there could be work done to fix these. The simulator is also lacking an automatic evaluation tool that does not require the tester to evaluate the cars performance visually. There is also the possibility to make the simulator more user friendly in the sense that objects, maps and vehicle parameters are changed with ease. Another future improvement of the simulator would be to create a random simulation generator that picks a route, places obstacles and other vehicles randomly along the route to simulate how the vehicle reacts.

Moreover, different scenarios and maps could be make such as intersections or curves. Apollo 2.5 have some new features compared to Apollo 2.0 these could be useful to improve the Twizys behavior, it would be interesting to learn how to use and implement the latest version.



## Chapter 11 BIBLIOGRAPHY

- [1] On-Road Automated Driving (ORAD) Committee, “SURFACE VEHICLE RECOMMENDED PRACTICE - Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles,” 2015.
- [2] “First self-driving bus in operation at Chalmers,” 2018. [Online].  
Available: <https://www.chalmers.se/en/areas-of-advance/Transport/news/Pages/First-self-driving-bus-in-operation-at-Chalmers.aspx>
- [3] Automotive News: [Online] Available: <http://nordic.businessinsider.com/google-self-driving-car-investment-exceeds-1-billion-2017-9?r=US&IR=T>
- [4] Waymo: [Online] Available: <https://waymo.com/>
- [5] Fred Lambert, Electrek: [Online] Available: <https://electrek.co/2017/12/08/elon-musk-tesla-self-driving-timeline/>
- [6] Automotive News: [Online] Available: <http://europe.autonews.com/article/20171212/ANE/171219914/volvos-drive-me-takes-detour-on-road-to-full-autonomy>
- [7] IEEE Spectrum: [Online] Available: <https://spectrum.ieee.org/cars-that-think/transportation/self-driving/nvidia-ceo-announces>
- [8] Tech Crunch: [Online] Available: <https://techcrunch.com/2017/06/26/volvo-and-autoliv-aim-to-sell-self-driving-cars-with-nvidia-ai-tech-by-2021/>
- [9] World Health Organization: [Online] Available: <http://www.who.int/mediacentre/factsheets/fs358/en/>
- [10] Apollo Auto: [Online] Available: <http://apollo.auto/fund.html>
- [11] Reuters Staff, Reuters: [Online] Available: <https://www.reuters.com/article/us-baidu-autonomous/baidu-receives-approval-from-california-dmv-to-test-self-driving-cars-idUSKCN11L2FU>
- [12] Kyle Hyatt, Road Show, by Cnet: [Online] Available: <https://www.cnet.com/roadshow/news/baidu-launches-version-2-0-of-its-apollo-self-driving-moonshot/>
- [13] Inductive charging: [Online] Available: <https://www.chalmers.se/SiteCollectionDocuments/E2/Kandidatprojekt/2018/EENX15-18-35.pdf>

- [14] Power Modelling for Energy Optimization: [Online] Available: <https://www.chalmers.se/SiteCollectionDocuments/E2/Kandidatprojekt/2018/EENX15-18-36.pdf>
- [15] Steering and motor control: [Online] Available: <https://www.chalmers.se/SiteCollectionDocuments/E2/Kandidatprojekt/2018/EENX15-18-37.pdf>
- [16] Safety: [Online] Available: <https://www.chalmers.se/SiteCollectionDocuments/E2/Kandidatprojekt/2018/EENX15-18-38.pdf>
- [17] Path Planning and Vehicle Control: [Online] Available: <https://www.chalmers.se/SiteCollectionDocuments/E2/Kandidatprojekt/2018/EENX15-18-35.pdf>
- [18] Perception: [Online] Available: <https://www.chalmers.se/SiteCollectionDocuments/E2/Kandidatprojekt/2018/EENX15-18-40.pdf>
- [19] Virtual integration and testing: [Online] Available: <https://www.chalmers.se/SiteCollectionDocuments/E2/Kandidatprojekt/2018/EENX15-18-41.pdf>
- [20] “Core Components,” p. 4, 2017. [Online]. Available: [http://www.ros.org/core-components/#communications\\_infrastructure](http://www.ros.org/core-components/#communications_infrastructure)
- [21] “ROS-master,” 2018. [Online]. Available: <http://wiki.ros.org/Master>
- [22] “ROS-command-line-tools,” 2015. [Online]. Available: <http://wiki.ros.org/ROS/CommandLineTools>
- [23] “ROS-roscap,” 2015. [Online]. Available: <http://wiki.ros.org/roscap>
- [24] “ROS-Message,” 2017. [Online]. Available: <http://wiki.ros.org/msg>
- [25] “Apollo Auto Dreamview,” 2017. [Online]. Available: [https://github.com/ApolloAuto/apollo/blob/r2.0.0/docs/specs/dreamview\\_usage\\_table.md](https://github.com/ApolloAuto/apollo/blob/r2.0.0/docs/specs/dreamview_usage_table.md)
- [26] I. Saito, “ROS-roscap-commandline,” 2018. [Online]. Available: <http://wiki.ros.org/roscap/CommandLine>
- [27] “Dreamview Usage Table.” [Online]. Available: [https://github.com/ApolloAuto/apollo/blob/r2.0.0/docs/specs/dreamview\\_usage\\_table.md](https://github.com/ApolloAuto/apollo/blob/r2.0.0/docs/specs/dreamview_usage_table.md)



- [28] “Video released of Uber self-driving crash that killed woman in Arizona.” [Online]. Available: <https://www.theguardian.com/technology/2018/mar/22/video-released-of-uber-self-driving-crash-that-killed-woman-in-arizona>
- [29] Software Integration and Simulation of Autonomous Vehicle Project Plan, Frigard Anton, 2018
- [30] Autonomous Twizy-Simulation using Apollo Auto, Frigard Anton, 2018
- [31] Baidu: [Online] Available: <https://baike.baidu.com/item/%E9%98%BF%E6%B3%A2%E7%BD%97/20625862>
- [32] United Nations: [Online] Available: <http://bigpicture.unfccc.int/#content-the-paris-agreemen>
- [33] European Commision: [Online] Available: [https://ec.europa.eu/clima/policies/transport/vehicles/cars\\_en](https://ec.europa.eu/clima/policies/transport/vehicles/cars_en)
- [34] «International Organization of Motor Vehicle Manufacturers» [Online] Available: <http://oica.net/wp-content/uploads/climate-change-and-co2-brochure.pdf>
- [35] «Self-Driving vehicles in logistics, DHL» [Online] Available: [http://www.dhl.com/content/dam/downloads/g0/about\\_us/logistics\\_insights/dhl\\_self\\_driving\\_vehicles.pdf](http://www.dhl.com/content/dam/downloads/g0/about_us/logistics_insights/dhl_self_driving_vehicles.pdf)
- [36] «Peak car Ownership» [Online] Available: [https://www.rmi.org/wp-content/uploads/2017/03/Mobility\\_PeakCarOwnership\\_Report2017.pdf](https://www.rmi.org/wp-content/uploads/2017/03/Mobility_PeakCarOwnership_Report2017.pdf)
- [37] [Online] Available: <https://www.ecosiglos.com/2014/12/los-beneficios-ambientales-de-los-vehiculos-autonomos.html>



# ***PART II BUDGET***



# Chapter 1 MEASUREMENTS

It must be said that all the material and project was done in Sweden. Therefore, all the budget will be calculated in Swedish krona.

---

## 1.1 *Hardware*

---

Reference	Element	Units
101	Embedded Computer	2
102	Nvidia GPU	2
103	SSD drive	2

---

## 1.2. *Software*

---

Reference	Programs/O.S.	Units
201	Virtual Box	2
202	Ubuntu	4
203	Apollo Auto	4

---

## 1.3. *Labor costs*

---

Reference	Tasks	Hours
301	Research	120
302	Use of software	500
303	Testing and problem solving	300
304	Report	130



## Chapter 2 UNIT PRICE

---

### 1.1 *Hardware*

---

Reference	Element	kSEK/unit
101	Embedded Computer	50
102	Nvidia GPU	30
103	SSD drive	5

---

### 2.2. *Software*

---

Reference	Programs/O.S.	kSEK/unit
201	Virtual Box	0
202	Ubuntu	0
203	Apollo Auto	0

---

### 2.3. *Labor costs*

---

Reference	Tasks	kSEK/hour
301	Research	40
302	Use of software	70
303	Testing and problem solving	60
304	Report	25





## Chapter 3 PARTIAL AMOUNT

---

### 3.1. *Hardware*

---

Reference	Element	Units	kSEK/unit	Total Amount (kSEK)
101	Embedded Computer	2	50	100
102	Nvidia GPU	2	30	60
103	SSD drive	4	5	20
<b>Total</b>				180

---

### 3.2. *Software*

---

Reference	Programs/O.S.	Units	kSEK/unit	Total Amount (kSEK)
201	Virtual Box	2	0	0
202	Ubuntu	4	0	0
203	Apollo Auto	4	0	0
<b>Total</b>				0

---

### 3.3. *Labor costs*

---

Reference	Tasks	Hours	kSEK/hour	Total Amount (kSEK)
301	Research	120	0.4	48
302	Use of software	500	0.7	350
303	Testing and problem solving	300	0.6	180
304	Report	130	0.25	32.5
<b>Total</b>				610.5



## Chapter 4 TOTAL AMOUNT

Section	Partial Amount(KSEK)
Hardware	180
Software	0
Labor costs	610.5
<b>Total</b>	<b>790.5</b>

A

Apollo publishing & subscribing  
structure

Module (Node)	Subscribing to	Publishing on
<b>Canbus</b>	/apollo/control	/apollo/canbus/chassis /apollo/canbus/chassis_detail
<b>Control</b>	/apollo/canbus/chassis /apollo/canbus/chassis_detail /apollo/planning /apollo/localization/msf_gnss /apollo/localization/msf_lidar /apollo/localization/msf_status /apollo/localization/pose	/apollo/control /apollo/control/pad
<b>Dreamview</b>	/apollo/localization/pose /apollo/canbus/chassis /apollo/canbus/chassis_detail /apollo/planning /apollo/monitor /apollo/monitor/static_info /apollo/perception/obstacles /apollo/prediction /apollo/routing_response	None
<b>Localization</b>	None	/apollo/localization/msf_gnss /apollo/localization/msf_lidar /apollo/localization/msf_status /apollo/localization/pose
<b>Map</b>	None	/apollo/relative_map /apollo/drive_event
<b>Monitor</b>	None	/apollo/monitor /apollo/monitor/static_info
<b>Perception</b>	None	/apollo/perception/obstacles /apollo/sensor/conti_radar /apollo/sensor/delphi_esr /apollo/perception/traffic_light /apollo/sensor/gnss/best_pose /apollo/sensor/gnss/corrected_imu /apollo/sensor/gnss/gnss_status /apollo/sensor/gnss/imu /apollo/sensor/gnss/ins_stat /apollo/sensor/gnss/odometry /apollo/sensor/gnss/rtk_eph /apollo/sensor/gnss/rtk_obs /apollo/sensor/mobileye
<b>Planning</b>	/apollo/perception/traffic_light /apollo/prediction /apollo/routing_response /apollo/canbus/chassis /apollo/canbus/chassis_detail /apollo/localization/msf_gnss /apollo/localization/msf_lidar /apollo/localization/msf_status /apollo/localization/pose	/apollo/planning /apollo/routing_request
<b>Prediction</b>	/apollo/perception/obstacles /apollo/perception/traffic_light	/apollo/prediction
<b>Routing</b>	/apollo/routing_request /apollo/monitor	/apollo/routing_response

# B

## base\_map.xml

### B.1 Map used in simulations

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <OpenDRIVE xmlns="http://www.opendrive.org">
3   <header
4     revMajor="1"
5     revMinor="0"
6     name="Chalmers HDMap"
7     version="1"
8     date="2018-04-10T09:35:00"
9     north="0"
10    south="0"
11    east="0"
12    west="0"
13    vendor="Autonomous Twizy"
14  >
15    <geoReference>
16      <![CDATA[+proj=utm +zone=31 +ellps=WGS84 +datum=WGS84 +units=m +no_defs]]>
17    </geoReference>
18  </header>
19  <road id="r1" junction="-1">
20    <link></link>
21    <objects>
22      <object id="1" type="crosswalk">
23        <outline>
24          <cornerGlobal x="40" y="0" z="0.00000000e+00" />
25          <cornerGlobal x="40" y="-6" z="0.00000000e+00" />
26          <cornerGlobal x="42" y="-6" z="0.00000000e+00" />
27          <cornerGlobal x="42" y="0" z="0.00000000e+00" />
28        </outline>
29      </object>
30    </objects>
31    <signals/>
32    <lanes>
33      <laneSection singleSide="true">
34        <boundaries>
35          <!-- Leftmost line containing the road (the reference line) -->
36          <boundary type="leftBoundary">
37            <geometry>
38              <pointSet>
39                <point x="0" y="0" z="0"/>
40                <point x="1000" y="0" z="0"/>
41              </pointSet>
42            </geometry>
43          </boundary>
44          <!-- Rightmost line containing the road (the rightmost lanes border) -->
45          <boundary type="rightBoundary">
46            <geometry>
```

```

47         <pointSet>
48             <point x="0" y="-6" z="0"/>
49             <point x="1000" y="-6" z="0"/>
50         </pointSet>
51     </geometry>
52 </boundary>
53 </boundaries>
54 <center>
55     <!-- reference line -->
56     <lane id="0" uid="r1_1_0" type="none">
57         <border>
58             <geometry sOffset="0" x="0" y="0" z="0" length="1000">
59                 <pointSet>
60                     <point x="0" y="0" z="0"/>
61                     <point x="1000" y="0" z="0"/>
62                 </pointSet>
63             </geometry>
64             <borderType sOffset="0" type="solid" color="yellow"/>
65         </border>
66     </lane>
67 </center>
68 <right>
69     <!-- First right lane -->
70     <lane id="-1" uid="r1_1_-1" type="driving" direction="forward" turnType="noTurn">
71         <centerLine> <!-- Lane center line -->
72             <geometry sOffset="0" x="0" y="-1.5" z="0" length="1000">
73                 <pointSet>
74                     <point x="0" y="-1.5" z="0"/>
75                     <point x="1000" y="-1.5" z="0"/>
76                 </pointSet>
77             </geometry>
78         </centerLine>
79         <border> <!-- Lane right side border -->
80             <geometry sOffset="0" x="0" y="-3" z="0" length="1000">
81                 <pointSet>
82                     <point x="0" y="-3" z="0"/>
83                     <point x="1000" y="-3" z="0"/>
84                 </pointSet>
85             </geometry>
86             <borderType sOffset="0" type="broken" color="white"/>
87         </border>
88         <link>
89             <neighbor id="r1_1_-2" side="right" direction="same" />
90         </link>
91         <speed max="50"/>
92         <sampleAssociates>
93             <sampleAssociate sOffset="0" leftWidth="1.5" rightWidth="1.5"/>
94             <sampleAssociate sOffset="1000" leftWidth="1.5" rightWidth="1.5"/>
95         </sampleAssociates>
96         <objectOverlapGroup>
97             <objectReference id="1" startOffset="40" endOffset="42"/>
98         </objectOverlapGroup>
99         <junctionOverlapGroup></junctionOverlapGroup>
100        <laneOverlapGroup>
101            <laneReference id="r1_1_-2" startOffset="0" endOffset="1000" isMerge="true"/>
102        </laneOverlapGroup>
103    </lane>
104    <!-- Second right lane -->
105    <lane id="-2" uid="r1_1_-2" type="driving" direction="forward" turnType="noTurn">
106        <centerLine> <!-- Lane center line -->
107            <geometry sOffset="0" x="0" y="-4.5" z="0" length="1000">

```

```

108         <pointSet>
109             <point x="0" y="-4.5" z="0"/>
110             <point x="1000" y="-4.5" z="0"/>
111         </pointSet>
112     </geometry>
113 </centerLine>
114 <border> <!-- Lane right side border -->
115     <geometry sOffset="0" x="0" y="-6" z="0" length="1000">
116         <pointSet>
117             <point x="0" y="-6" z="0"/>
118             <point x="1000" y="-6" z="0"/>
119         </pointSet>
120     </geometry>
121     <borderType sOffset="0" type="broken" color="white"/>
122 </border>
123 <link>
124     <neighbor id="r1_1_-1" side="left" direction="same" />
125 </link>
126 <speed max="50"/>
127 <sampleAssociates>
128     <sampleAssociate sOffset="0" leftWidth="1.5" rightWidth="1.5"/>
129     <sampleAssociate sOffset="1000" leftWidth="1.5" rightWidth="1.5"/>
130 </sampleAssociates>
131 <objectOverlapGroup>
132     <objectReference id="1" startOffset="40" endOffset="42" />
133 </objectOverlapGroup>
134 <junctionOverlapGroup></junctionOverlapGroup>
135 <laneOverlapGroup>
136     <laneReference id="r1_1_-1" startOffset="0" endOffset="1000" isMerge="true"/>
137 </laneOverlapGroup>
138 </lane>
139 </right>
140 </laneSection>
141 </lanes>
142 </road>
143 </OpenDRIVE>

```

## B.2 Map used in test drives

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3   https://www.maps.ie/coordinates.html
4   https://pastebin.com/pBRHXRgk
5 -->
6 <OpenDRIVE xmlns="http://www.opendrive.org">
7   <header
8     revMajor="1"
9     revMinor="0"
10    name="Chalmers HDMaP"
11    version="1"
12    date="2018-04-10T09:35:00"
13    north="57.69004972576799"
14    south="57.686929994328885"
15    east="11.982407065449252"
16    west="11.976828070698275"
17    vendor="Perception Squad"
18  >
19   <geoReference>
20     <![CDATA[+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs]]>
21   </geoReference>

```



```
22 </header>
23 <road id="r1" junction="-1">
24   <link>
25     <!--
26     <predecessor elementType="road" elementId="nât" contactPoint="start"/>
27     <successor elementType="road" elementId="nât" contactPoint="end"/>
28     -->
29   </link>
30   <objects/>
31   <signals/>
32   <lanes>
33     <laneSection singleSide="true">
34       <boundaries>
35         <!-- Leftmost line containing the road (the reference line) -->
36         <boundary type="leftBoundary">
37           <geometry>
38             <pointSet>
39               <point x="11.9777109467941" y="57.68812039715326" z="0"/>
40               <point x="11.977468634345016" y="57.688394602302274" z="0"/>
41               <point x="11.97744176754415" y="57.68843741455268" z="0"/>
42               <point x="11.977068370245743" y="57.6888556551523" z="0"/>
43             </pointSet>
44           </geometry>
45         </boundary>
46         <!-- Rightmost line containing the road (the rightmost lanes border) -->
47         <boundary type="rightBoundary">
48           <geometry>
49             <pointSet>
50               <point x="11.9777397547" y="57.6881278225" z="0"/>
51               <point x="11.9774926684" y="57.6884021175" z="0"/>
52               <point x="11.9774678886" y="57.6884455305" z="0"/>
53               <point x="11.9770924222" y="57.6888620698" z="0"/>
54             </pointSet>
55           </geometry>
56         </boundary>
57       </boundaries>
58       <center>
59         <!-- reference line -->
60         <lane id="0" uid="r1_1_0" type="none">
61           <border>
62             <geometry sOffset="0" x="11.9777109467941" y="57.68812039715326" z="0" length="90.1586">
63               <pointSet>
64                 <point x="11.9777109467941" y="57.68812039715326" z="0"/>
65                 <point x="11.977468634345016" y="57.688394602302274" z="0"/>
66                 <point x="11.97744176754415" y="57.68843741455268" z="0"/>
67                 <point x="11.977068370245743" y="57.6888556551523" z="0"/>
68               </pointSet>
69             </geometry>
70             <borderType sOffset="0" type="broken" color="white"/>
71           </border>
72         </lane>
73       </center>
74       <right>
75         <!-- First right lane -->
76         <lane id="-1" uid="r1_1_-1" type="driving" direction="forward" turnType="noTurn">
77           <centerLine> <!-- Lane center line -->
78             <geometry sOffset="0" y="11.977725350745573" x="57.68812410982745" z="0" length="90.1586">
79               <pointSet>
80                 <point x="11.977725350745573" y="57.68812410982745" z="0"/>
81                 <point x="11.97748065137126" y="57.688398359901704" z="0"/>
82                 <point x="11.977454828070613" y="57.68844147252701" z="0"/>

```

```

83         <point x="11.977080396221808" y="57.68885886247673" z="0"/>
84     </pointSet>
85 </geometry>
86 </centerLine>
87 <border> <!-- Lane right side border -->
88     <geometry sOffset="0" x="11.9777397547" y="57.6881278225" z="0" length="90.1586">
89         <pointSet>
90             <point x="11.9777397547" y="57.6881278225" z="0"/>
91             <point x="11.9774926684" y="57.6884021175" z="0"/>
92             <point x="11.9774678886" y="57.6884455305" z="0"/>
93             <point x="11.9770924222" y="57.6888620698" z="0"/>
94         </pointSet>
95     </geometry>
96     <borderType sOffset="0" type="broken" color="white"/>
97 </border>
98 <link>
99     <!--
100     <predecessor id=""/>
101     <successor id=""/>
102     -->
103 </link>
104 <speed max="5"/>
105 <sampleAssociates>
106     <sampleAssociate sOffset="0.0" leftWidth="1.5" rightWidth="1.5"/>
107     <sampleAssociate sOffset="33.6782" leftWidth="1.5" rightWidth="1.5"/>
108     <sampleAssociate sOffset="38.6931" leftWidth="1.5" rightWidth="1.5"/>
109     <sampleAssociate sOffset="90.1586" leftWidth="1.5" rightWidth="1.5"/>
110 </sampleAssociates>
111 <junctionOverlapGroup></junctionOverlapGroup>
112 <laneOverlapGroup></laneOverlapGroup>
113 </lane>
114 </right>
115 </laneSection>
116 </lanes>
117 </road>
118 <road id="r2" junction="-1">
119     <link>
120         <!--
121         <predecessor elementType="road" elementId="nât" contactPoint="start"/>
122         <successor elementType="road" elementId="nât" contactPoint="end"/>
123         -->
124     </link>
125 <objects/>
126 <signals/>
127 <lanes>
128     <laneSection singleSide="true">
129         <boundaries>
130             <!-- Leftmost line containing the road (the reference line) -->
131             <boundary type="leftBoundary">
132                 <geometry>
133                     <pointSet>
134                         <point x="11.977068370245743" y="57.6888556551523" z="0"/>
135                         <point x="11.97744176754415" y="57.68843741455268" z="0"/>
136                         <point x="11.977468634345016" y="57.688394602302274" z="0"/>
137                         <point x="11.9777109467941" y="57.68812039715326" z="0"/>
138                     </pointSet>
139                 </geometry>
140             </boundary>
141             <!-- Rightmost line containing the road (the rightmost lanes border) -->
142             <boundary type="rightBoundary">
143                 <geometry>

```

```
144         <pointSet>
145             <point x="11.9770443183" y="57.6888492405" z="0"/>
146             <point x="11.9774156465" y="57.6884292986" z="0"/>
147             <point x="11.9774446003" y="57.6883870871" z="0"/>
148             <point x="11.9776821389" y="57.6881129718" z="0"/>
149         </pointSet>
150     </geometry>
151 </boundary>
152 </boundaries>
153 <center>
154     <!-- reference line -->
155     <lane id="0" uid="r2_1_0" type="none">
156         <border>
157             <geometry sOffset="0" x="11.977068370245743" y="57.6888556551523" z="0" length="90.1586">
158                 <pointSet>
159                     <point x="11.977068370245743" y="57.6888556551523" z="0"/>
160                     <point x="11.97744176754415" y="57.68843741455268" z="0"/>
161                     <point x="11.977468634345016" y="57.688394602302274" z="0"/>
162                     <point x="11.9777109467941" y="57.68812039715326" z="0"/>
163                 </pointSet>
164             </geometry>
165             <borderType sOffset="0" type="broken" color="white"/>
166         </border>
167     </lane>
168 </center>
169 <right>
170     <!-- First right lane -->
171     <lane id="-1" uid="r2_1_-1" type="driving" direction="forward" turnType="noTurn">
172         <centerLine> <!-- Lane center line -->
173             <geometry sOffset="0" y="11.977056344271809" x="57.688852447826726" z="0" length="90.1586">
174                 <pointSet>
175                     <point x="11.977056344271809" y="57.688852447826726" z="0"/>
176                     <point x="11.977428707020614" y="57.68843335657701" z="0"/>
177                     <point x="11.977456617321263" y="57.68839084470171" z="0"/>
178                     <point x="11.977696542845573" y="57.68811668447744" z="0"/>
179                 </pointSet>
180             </geometry>
181         </centerLine>
182         <border> <!-- Lane right side border -->
183             <geometry sOffset="0" x="11.9777397547" y="57.6881278225" z="0" length="90.1586">
184                 <pointSet>
185                     <point x="11.9770443183" y="57.6888492405" z="0"/>
186                     <point x="11.9774156465" y="57.6884292986" z="0"/>
187                     <point x="11.9774446003" y="57.6883870871" z="0"/>
188                     <point x="11.9776821389" y="57.6881129718" z="0"/>
189                 </pointSet>
190             </geometry>
191             <borderType sOffset="0" type="broken" color="white"/>
192         </border>
193         <link>
194             <!--
195             <predecessor id=""/>
196             <successor id=""/>
197             -->
198         </link>
199         <speed max="5"/>
200         <sampleAssociates>
201             <sampleAssociate sOffset="0" leftWidth="1.5" rightWidth="1.5"/>
202             <sampleAssociate sOffset="51.4655" leftWidth="1.5" rightWidth="1.5"/>
203             <sampleAssociate sOffset="56.4804" leftWidth="1.5" rightWidth="1.5"/>
204             <sampleAssociate sOffset="90.1586" leftWidth="1.5" rightWidth="1.5"/>

```

```
205         </sampleAssociates>
206         <junctionOverlapGroup></junctionOverlapGroup>
207         <laneOverlapGroup></laneOverlapGroup>
208     </lane>
209 </right>
210 </laneSection>
211 </lanes>
212 </road>
213 </OpenDRIVE>
```



# C

## Scripts for simulating obstacles

### C.1 sim\_test.cc

```
1 #include "modules/dreamview/backend/sim_control/sim_test.h"
2
3 #include <cmath>
4
5 #include "modules/common/math/math_utils.h"
6 #include "modules/common/math/quaternion.h"
7 #include "modules/common/time/time.h"
8 #include "modules/common/util/file.h"
9
10 //ERIK (some might be unnecessary)
11 #include "modules/common/adapters/adapter_gflags.h"
12 #include "modules/dreamview/backend/common/dreamview_gflags.h"
13 #include "modules/common/adapters/proto/adapter_config.pb.h"
14 #include "modules/common/util/util.h"
15 #include "modules/common/adapters/adapter_manager.h"
16
17 namespace apollo {
18   namespace dreamview {
19
20     using apollo::common::adapter::PerceptionObstaclesAdapter;
21     using apollo::perception::Point;
22     using apollo::perception::PerceptionObstacle;
23     using apollo::perception::PerceptionObstacles;
24
25     double y;
26     double x;
27     double x_change;
28     double y_change;
29     bool dec;
30
31
32     void updateSimWorld(){
33
34       PerceptionObstacles obstacles; //creates an obstacles which is a protobuff msg that
35         will contain obstacle(s)
36       //Declares global variables
37       apollo::common::adapter::AdapterManager::FillPerceptionObstaclesHeader("perception_obstacle",
38         &obstacles); //Updates the timestamps
39       if (dec != true){
40         x =20;
41         y=-1.5;
42         x_change=0.01;
43         y_change=0.004;
44         dec = true;
45       }
46     }
47   }
48 }
```

```

45     // Scenario 1 - Pedestrian crossing (comment out while running other
        scenarios)
46     addPerceptionObstacle(&obstacles,
        ::apollo::perception::PerceptionObstacle::PEDESTRIAN, 1, 41, y, 0, 0, 1, 0,
        1, 1, 1);
47     if(y<-6.5){
48         y=0.5;
49     }else{
50         y=y-y_change;
51     }
52
53
54     //Scenario 2 - Car chase (comment out while running other scenarios)
55     addPerceptionObstacle(&obstacles, ::apollo::perception::PerceptionObstacle::VEHICLE,
        1, x, y, 0, sqrt(pow(x_change,2)+pow(y_change,2)),
        sqrt(pow(x_change,2)+pow(y_change,2)), 0, 1, 0.5, 1);
56     x = x+x_change;
57     if(x>30 && y>-4.5){
58         x=x+x_change;
59         y=y-y_change;
60     }
61     if(x>80){
62         x=20;
63         y=-1.5;
64     }
65
66
67     /* How obstacles are made: adds static obstacle(s) to obstacles, what is sent
        is: a pointer to the obstacles that the obstacle should be added to, what
        type of obstacle should be added, the id of the obstacle,the x-coordinate to
        the centrum of the obstacle, y- -||-, z- -||-, width of the obstacle(x),
        length of the obstacle(y), height of the obstacle(z)
68     */
69
70     //publish the msg
71     apollo::common::adapter::AdapterManager::PublishPerceptionObstacles(obstacles);
72 }
73
74     /* Adds obstacle to obstacles, what is requerd: a pointer to the obstacles that
        the obsticle should be added to, what type of obstacle should be added, the id
        of the obstacle, the x-coordinat to centrum of the obstacle, y- -||-,
        z-coordinate to the bootom of the obstacle, ith of the obstacle(x), legnth of
        the obstacle(y), height of the obstacle(z)
75     */
76     void addPerceptionObstacle(perception::PerceptionObstacles *obstacles,
        perception::PerceptionObstacle::Type type, int id, double Xcoord, double Ycoord,
        double Zcoord, double width, double length, double height){
77     //Calls on addPerceptionObstacle with speed set to 0 in x,y,z
78     addPerceptionObstacle(obstacles, type, id, Xcoord, Ycoord, Zcoord, 0, 0, 0, width,
        length, height);
79 }
80
81     /* Adds obstacle to obstacles, what is requerd: a pointer to the obstacles that the
        obsticle should be added to, what type of obstacle should be added, the id of the
        obstacle, the x-coordinat to centrum of the obstacle, y- -||-, z-coordinate to the
        bootom of the obstacle, ith of the obstacle(x), legnth of the obstacle(y), height of
        the obstacle(z)
82     */

```

```

83 void addPerceptionObstacle(perception::PerceptionObstacles *obstacles,
    perception::PerceptionObstacle::Type type, int id, double Xcoord, double Ycoord,
    double Zcoord, double xVel, double yVel, double zVel, double width, double length,
    double height){
84 //adds an obstacle called obst to obstacles
85 auto obst = obstacles->add_perception_obstacle();
86
87 obst->set_id(id);
88 obst->set_length(length);
89 obst->set_width(width);
90 obst->set_height(height);
91 obst->set_type(type);
92
93 auto pointy = obst->mutable_position();
94 auto pointy2 = obst->mutable_velocity();
95
96 pointy->set_x(Xcoord);
97 pointy->set_y(Ycoord);
98 pointy->set_z(Zcoord);
99
100 pointy2->set_x(xVel);
101 pointy2->set_y(yVel);
102 pointy2->set_z(zVel);
103
104 //Adds a polygone point which are the corners of the obstacle
105 auto pen0 = obst->add_polygon_point();
106 auto pen1 = obst->add_polygon_point();
107 auto pen2 = obst->add_polygon_point();
108 auto pen3 = obst->add_polygon_point();
109 auto pen4 = obst->add_polygon_point();
110 auto pen5 = obst->add_polygon_point();
111 auto pen6 = obst->add_polygon_point();
112 auto pen7 = obst->add_polygon_point();
113
114 //Makes a box round the point where the obstacle is, the the point is on the
    center of the "flor" of the box
115 pen0->set_x(Xcoord-width/2);
116 pen0->set_y(Ycoord-length/2);
117 pen0->set_z(Zcoord);
118 pen1->set_x(Xcoord+width/2);
119 pen1->set_y(Ycoord-length/2);
120 pen1->set_z(Zcoord);
121 pen2->set_x(Xcoord+width/2);
122 pen2->set_y(Ycoord+length/2);
123 pen2->set_z(Zcoord);
124 pen3->set_x(Xcoord-width/2);
125 pen3->set_y(Ycoord+length/2);
126 pen3->set_z(Zcoord);
127 pen4->set_x(Xcoord-width/2);
128 pen4->set_y(Ycoord-length/2);
129 pen4->set_z(Zcoord+height);
130 pen5->set_x(Xcoord+width/2);
131 pen5->set_y(Ycoord-length/2);
132 pen5->set_z(Zcoord+height);
133 pen6->set_x(Xcoord+width/2);
134 pen6->set_y(Ycoord+length/2);
135 pen6->set_z(Zcoord+height);
136 pen7->set_x(Xcoord-width/2);
137 pen7->set_y(Ycoord+length/2);
138 pen7->set_z(Zcoord+height);
139

```



```

140     }
141   }
142
143
144 }
145 }

```

## C.2 sim\_test.h

```

1 #ifndef MODULES_DREAMVIEW_BACKEND_SIM_CONTROL_TEST_H_
2 #define MODULES_DREAMVIEW_BACKEND_SIM_CONTROL_TEST_H_
3
4 #include <string>
5
6 #include "gtest/gtest_prod.h"
7 #include "modules/common/adapters/adapter_manager.h"
8 #include "modules/dreamview/backend/common/dreamview_gflags.h"
9 #include "modules/dreamview/backend/map/map_service.h"
10
11
12 /**
13  * @namespace apollo::dreamview
14  * @brief apollo::dreamview
15  */
16 namespace apollo {
17   namespace dreamview {
18
19     void updateSimWorld();
20
21     void addPerceptionObstacle(perception::PerceptionObstacles *obstacles,
22                               perception::PerceptionObstacle::Type type, int id, double Xcoord,
23                               double Ycoord, double Zcoord, double xVel, double yVel, double zVel,
24                               double width, double length, double height);
25
26     void addPerceptionObstacle(perception::PerceptionObstacles *obstacles,
27                               perception::PerceptionObstacle::Type type, int id, double Xcoord,
28                               double Ycoord, double Zcoord, double width, double length, double height);
29
30
31   }
32 }
33
34 #endif

```

## C.3 sim\_control.cc

```

1 /*****
2  * Copyright 2017 The Apollo Authors. All Rights Reserved.
3  *
4  * Licensed under the Apache License, Version 2.0 (the "License");
5  * you may not use this file except in compliance with the License.
6  * You may obtain a copy of the License at
7  *
8  * http://www.apache.org/licenses/LICENSE-2.0
9  *
10 * Unless required by applicable law or agreed to in writing, software
11 * distributed under the License is distributed on an "AS IS" BASIS,
12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 * See the License for the specific language governing permissions and

```

```

14  * limitations under the License.
15  *****/
16
17 #include "modules/dreamview/backend/sim_control/sim_control.h"
18
19 #include <cmath>
20
21 #include "modules/common/math/math_utils.h"
22 #include "modules/common/math/quaternion.h"
23 #include "modules/common/time/time.h"
24 #include "modules/common/util/file.h"
25
26 //ERIK
27 #include "modules/dreamview/backend/sim_control/sim_test.h"
28
29 namespace apollo {
30 namespace dreamview {
31
32 using apollo::canbus::Chassis;
33 using apollo::common::Point3D;
34 using apollo::common::Quaternion;
35 using apollo::common::TrajectoryPoint;
36 using apollo::common::adapter::AdapterManager;
37 using apollo::common::Header;
38 using apollo::common::math::HeadingToQuaternion;
39 using apollo::common::math::InverseQuaternionRotate;
40 using apollo::common::math::NormalizeAngle;
41 using apollo::common::math::QuaternionToHeading;
42 using apollo::common::time::Clock;
43 using apollo::common::util::GetProtoFromFile;
44 using apollo::localization::LocalizationEstimate;
45 using apollo::routing::RoutingResponse;
46
47
48
49
50
51 namespace {
52
53 void TransformToVRF(const Point3D& point_mrf, const Quaternion& orientation,
54                   Point3D* point_vrf) {
55     Eigen::Vector3d v_mrf(point_mrf.x(), point_mrf.y(), point_mrf.z());
56     auto v_vrf = InverseQuaternionRotate(orientation, v_mrf);
57     point_vrf->set_x(v_vrf.x());
58     point_vrf->set_y(v_vrf.y());
59     point_vrf->set_z(v_vrf.z());
60 }
61
62 bool CompareHeader(const Header& lhs, const Header& rhs) {
63     return lhs.sequence_num() == rhs.sequence_num() &&
64            lhs.timestamp_sec() == rhs.timestamp_sec();
65 }
66
67 } // namespace
68
69 SimControl::SimControl(const MapService* map_service)
70     : map_service_(map_service),
71       prev_point_index_(0),
72       next_point_index_(0),
73       received_planning_(false),
74       planning_count_(-1),

```

```
75     re_routing_triggered_(false),
76     enabled_(FLAGS_enable_sim_control) {}
77
78 void SimControl::Init(bool set_start_point, double start_velocity,
79                     double start_acceleration) {
80     // Setup planning and routing result data callback.
81
82     AdapterManager::AddPlanningCallback(&SimControl::OnPlanning, this);
83     AdapterManager::AddRoutingResponseCallback(&SimControl::OnRoutingResponse,
84                                               this);
85
86     // Start timer to publish localization and chassis messages.
87     sim_control_timer_ = AdapterManager::CreateTimer(
88         ros::Duration(kSimControlInterval), &SimControl::TimerCallback, this);
89
90     if (set_start_point) {
91         apollo::common::PointENU start_point;
92         if (!map_service_->GetStartPoint(&start_point)) {
93             AWARN << "Failed to get a dummy start point from map!";
94             return;
95         }
96         SetStartPoint(start_point.x(), start_point.y());
97     }
98
99     start_velocity_ = start_velocity;
100    start_acceleration_ = start_acceleration;
101 }
102
103 void SimControl::SetStartPoint(const double x, const double y) {
104     next_point_.set_v(start_velocity_);
105     next_point_.set_a(start_acceleration_);
106
107     auto* next_point = next_point_.mutable_path_point();
108     next_point->set_x(x);
109     next_point->set_y(y);
110     next_point->set_z(0.0);
111
112     double theta = 0.0;
113     double s = 0.0;
114     if (!map_service_->GetPoseWithRegardToLane(next_point->x(), next_point->y(),
115                                               &theta, &s)) {
116         AERROR << "Failed to get heading from map! Treat theta and s as 0.0!";
117     }
118     next_point->set_theta(theta);
119     next_point->set_s(s);
120     next_point->set_kappa(0.0);
121
122     prev_point_index_ = next_point_index_ = 0;
123     received_planning_ = false;
124
125     Start();
126 }
127
128 void SimControl::ClearPlanning() {
129     current_trajectory_.Clear();
130     received_planning_ = false;
131     planning_count_ = 0;
132 }
133
134 void SimControl::OnRoutingResponse(const RoutingResponse& routing) {
135     CHECK_LE(2, routing.routing_request().waypoint_size());
```

```

136     const auto& start_pose = routing.routing_request().waypoint(0).pose();
137
138     current_routing_header_ = routing.header();
139
140     // If this is from a planning re-routing request, don't reset car's location.
141     re_routing_triggered_ =
142         routing.routing_request().header().module_name() == "planning";
143     if (!re_routing_triggered_) {
144         ClearPlanning();
145         SetStartPoint(start_pose.x(), start_pose.y());
146     }
147 }
148
149 void SimControl::Start() {
150     if (enabled_) {
151         sim_control_timer_.start();
152     }
153 }
154
155 void SimControl::Stop() { sim_control_timer_.stop(); }
156
157 void SimControl::OnPlanning(const apollo::planning::ADCTrajectory& trajectory) {
158     // Reset current trajectory and the indices upon receiving a new trajectory.
159     // The routing SimControl owns must match with the one Planning has.
160     if (re_routing_triggered_ ||
161         CompareHeader(trajectory.routing_header(), current_routing_header_)) {
162         // Hold a few cycles until the position information is fully refreshed on
163         // planning side. Don't wait for the very first planning received.
164         ++planning_count_;
165         if (planning_count_ == 0 || planning_count_ >= kPlanningCountToStart) {
166             planning_count_ = kPlanningCountToStart;
167             current_trajectory_ = trajectory;
168             prev_point_index_ = 0;
169             next_point_index_ = 0;
170             received_planning_ = true;
171         }
172     } else {
173         ClearPlanning();
174     }
175 }
176
177 void SimControl::Freeze() {
178     next_point_.set_v(0.0);
179     next_point_.set_a(0.0);
180     prev_point_ = next_point_;
181 }
182
183 double SimControl::AbsoluteTimeOfNextPoint() {
184     return current_trajectory_.header().timestamp_sec() +
185         current_trajectory_.trajectory_point(next_point_index_)
186             .relative_time();
187 }
188
189 bool SimControl::NextPointWithinRange() {
190     return next_point_index_ < current_trajectory_.trajectory_point_size() - 1;
191 }
192
193 void SimControl::TimerCallback(const ros::TimerEvent& event) { RunOnce(); }
194
195 void SimControl::RunOnce() {
196     // Result of the interpolation.

```

```
197 double lambda = 0.0;
198 auto current_time = Clock::NowInSeconds();
199
200 if (!received_planning_) {
201     prev_point_ = next_point_;
202 } else {
203     if (current_trajectory_.estop().is_estop() || !NextPointWithinRange()) {
204         // Freeze the car when there's an estop or the current trajectory has been
205         // exhausted.
206         Freeze();
207     } else {
208         // Determine the status of the car based on received planning message.
209         double timestamp = current_trajectory_.header().timestamp_sec();
210
211         while (NextPointWithinRange() &&
212              current_time > AbsoluteTimeOfNextPoint()) {
213             ++next_point_index_;
214         }
215
216         if (next_point_index_ == 0) {
217             AERROR << "First trajectory point is a future point!";
218             return;
219         }
220
221         if (current_time > AbsoluteTimeOfNextPoint()) {
222             prev_point_index_ = next_point_index_;
223         } else {
224             prev_point_index_ = next_point_index_ - 1;
225         }
226
227         next_point_ = current_trajectory_.trajectory_point(next_point_index_);
228         prev_point_ = current_trajectory_.trajectory_point(prev_point_index_);
229
230         // Calculate the ratio based on the position of current time in
231         // between the previous point and the next point, where lambda =
232         // (current_point - prev_point) / (next_point - prev_point).
233         if (next_point_index_ != prev_point_index_) {
234             lambda = (current_time - timestamp - prev_point_.relative_time()) /
235                    (next_point_.relative_time() - prev_point_.relative_time());
236         }
237     }
238 }
239
240 PublishChassis(lambda);
241 PublishLocalization(lambda);
242 }
243
244
245
246 void SimControl::PublishChassis(double lambda) {
247     Chassis chassis;
248     AdapterManager::FillChassisHeader("SimControl", &chassis);
249
250     chassis.set_engine_started(true);
251     chassis.set_driving_mode(Chassis::COMPLETE_AUTO_DRIVE);
252     chassis.set_gear_location(Chassis::GEAR_DRIVE);
253
254     double cur_speed = Interpolate(prev_point_.v(), next_point_.v(), lambda);
255     chassis.set_speed_mps(cur_speed);
256     chassis.set_throttle_percentage(0.0);
257     chassis.set_brake_percentage(0.0);
```

```

258
259     AdapterManager::PublishChassis(chassis);
260 }
261
262 void SimControl::PublishLocalization(double lambda) {
263     LocalizationEstimate localization;
264     AdapterManager::FillLocalizationHeader("SimControl", &localization);
265
266     auto* pose = localization.mutable_pose();
267     auto prev = prev_point_.path_point();
268     auto next = next_point_.path_point();
269
270     // Set position
271     double cur_x = Interpolate(prev.x(), next.x(), lambda);
272     pose->mutable_position()->set_x(cur_x);
273     double cur_y = Interpolate(prev.y(), next.y(), lambda);
274     pose->mutable_position()->set_y(cur_y);
275     double cur_z = Interpolate(prev.z(), next.z(), lambda);
276     pose->mutable_position()->set_z(cur_z);
277
278     // Set orientation and heading
279     double cur_theta = NormalizeAngle(
280         prev.theta() + lambda * NormalizeAngle(next.theta() - prev.theta()));
281
282     Eigen::Quaternion<double> cur_orientation =
283         HeadingToQuaternion<double>(cur_theta);
284     pose->mutable_orientation()->set_qw(cur_orientation.w());
285     pose->mutable_orientation()->set_qx(cur_orientation.x());
286     pose->mutable_orientation()->set_qy(cur_orientation.y());
287     pose->mutable_orientation()->set_qz(cur_orientation.z());
288     pose->set_heading(cur_theta);
289
290     // Set linear_velocity
291     double cur_speed = Interpolate(prev_point_.v(), next_point_.v(), lambda);
292     pose->mutable_linear_velocity()->set_x(std::cos(cur_theta) * cur_speed);
293     pose->mutable_linear_velocity()->set_y(std::sin(cur_theta) * cur_speed);
294     pose->mutable_linear_velocity()->set_z(0);
295
296     // Set angular_velocity in both map reference frame and vehicle reference
297     // frame
298     double cur_curvature = Interpolate(prev.kappa(), next.kappa(), lambda);
299     pose->mutable_angular_velocity()->set_x(0);
300     pose->mutable_angular_velocity()->set_y(0);
301     pose->mutable_angular_velocity()->set_z(cur_speed * cur_curvature);
302
303     TransformToVRF(pose->angular_velocity(), pose->orientation(),
304         pose->mutable_angular_velocity_vrf());
305
306     // Set linear_acceleration in both map reference frame and vehicle reference
307     // frame
308     double cur_acceleration_s =
309         Interpolate(prev_point_.a(), next_point_.a(), lambda);
310     auto* linear_acceleration = pose->mutable_linear_acceleration();
311     linear_acceleration->set_x(std::cos(cur_theta) * cur_acceleration_s);
312     linear_acceleration->set_y(std::sin(cur_theta) * cur_acceleration_s);
313     linear_acceleration->set_z(0);
314
315     TransformToVRF(pose->linear_acceleration(), pose->orientation(),
316         pose->mutable_linear_acceleration_vrf());
317
318

```

```
319
320
321   AdapterManager::PublishLocalization(localization);
322
323   //ERIK
324   //Calls the code that generates obstacles
325   updateSimWorld();
326
327
328 }
329
330
331
332 } // namespace dreamview
333 } // namespace apollo
```

# E

## How-to document

This document contains detailed information on how to setup Apollo Auto and how to customize it. It should be viewed as a collection of text written by the Apollo Auto team and some from experience of the authors. Therefore a lot of credit from this document should go to the Apollo Auto team as well as the authors of this thesis. For further information on Apollo Auto visit <https://github.com/ApolloAuto>.



# Contents

<b>1</b>	<b>Computer setup</b>	<b>1</b>
1.1	Hardware . . . . .	1
1.2	Software . . . . .	1
1.2.1	Technical installation process . . . . .	2
<b>2</b>	<b>Bag-file Simulation</b>	<b>4</b>
2.1	Creating bag-files . . . . .	4
<b>3</b>	<b>Complete system simulation</b>	<b>5</b>
3.1	The three maps essential for simulation . . . . .	5
3.2	Creating a new map . . . . .	5
3.3	Changing maps . . . . .	6
3.4	Implement CAN for new vehicle . . . . .	6
3.5	Starting a new simulation . . . . .	7
3.6	Adding obstacles to the simulation . . . . .	7
3.7	Debugging . . . . .	8
<b>4</b>	<b>CAN</b>	<b>9</b>
<b>5</b>	<b>Add an Adapter</b>	<b>10</b>

# Computer setup

This chapter explains how the computer(Nuvo-6108-GC, hereby referred to as Novu) is set-up properly for use with Apollo Auto. Further information on how the set-up process is conducted can be found on the Apollo Auto github.

## 1.1 Hardware

Firstly the SSD, GPU and CAN-card are installed in the case. Remember to screw each component into the case with the existing holders, this is to remove risk of the contacts breaking from bumps and such while driving. Observe that the CAN-card does not have an extra holder, but is only held in place with one screw.

Something to note about installing the CAN-card is that we had to bend the outer shell of the case a bit with our hands to make it fit. Do not press excessively but just a little ( $< .5$  mm).

The graphics card is placed on the PCI-express x16 port that is located as far away from the heat sink. Make sure that the holder is fastened as well, it is a small metal holder that fits on the top of the graphics card.

## 1.2 Software

Firstly Ubuntu has to be installed. Download it from their website (<https://www.ubuntu.com>), as of 27/2-18 Apollo runs on Ubuntu 14.04 with kernel "linux-4.4.32-apollo-1.0.0".

To be able to install Ubuntu the USB-drive has to be bootable, with some quick searches on Google this is easy to find and do. One example of a program that does this is Rufus (<https://rufus.akeo.ie>).

When Ubuntu is installed it is recommended to turn the fan speed up. This is done by restarting the computer and holding down the F2 button. When you get to the BIOS go to "Advanced", "Smart Fan Setting". Then changed "Max" to 50 degrees and "Start" to 20.

The installation process for the Apollo-kernel will not be described in detail here but can be found on the Apollo Auto Github as read-me files. The kernel can be downloaded from the same place (<https://github.com/ApolloAuto/apollo-kernel>).

To check if the correct kernel is running use the command:

```
uname -r
```

## 1.2.1 Technical installation process

Copy the ''esdcan .....

```
> Write the ISO-file to the USB-drive, with for example Rufus.  
// Done on a separate computer
```

```
> Start the Novu, with the USB connected, hold F12 during the start up.  
//The BIOS is entered by holding F12  
> When in the BIOS, choose to boot from USB.  
> Follow the installation guide and install Ubuntu.
```

```
> When booted up, open the terminal.  
> Run "apollosetup1.sh"
```

```
> Reboot the Novu
```

```
> Run apollosetup2.sh  
// This might not work correctly the follow the next two steps.  
// Copy the git clone by hand and run the apollosetup2.sh again:  
// > git clone https://https://github.com/ApolloAuto/apollo.git
```

```
> Copy ntacn.h (located in esdcan/include) to  
apollo/third_party/can_card_library/esd_can/include.
```

```
> Copy libntacn.so.4.0.1 (located in lib64) to  
apollo/third_party/can_card_library/esd_can/lib
```

NOTE! You have to create the final maps ''include'' and ''lib'' by yourself.

```
> In apollo/third_party/can_card_library/esd_can run:  
  > cd ./lib/;  
  > ln -s libntcan.so.4.0.1 libntcan.so.4;  
  > ln -s libntcan.so.4.0.1 libntcan.so.4.0
```

## apollosetup1.txt

---

```
#!/bin/bash

sudo apt-get update; sudo apt-get upgrade -Y

sudo apt-get install -Y linux-generic-lts-xenial

tar zxvf linux-4.4.32-apollo-1.5.0.tar.gz
cd install
sudo bash install_kernel.sh -Y

echo "Please reboot in apollo-kernel"
```

---

## apollosetup2.txt

---

```
#!/bin/bash

cd esdcan-pcie402-linux-2.6.x-x86_64-3.10.4

cd src/; make -C /lib/modules/$(uname -r)/build M='pwd'
sudo make -C /lib/modules/$(uname -r)/build M='pwd' modules_install
cd ../../

sudo apt install git

git clone https://github.com/ApolloAuto/apollo.git
cd apollo
git checkout r2.0.0; cd apollo;

echo "export APOLLO_HOME=$(pwd)" >> ~/.bashrc && source ~/.bashrc
source ~/.bashrc

cd $APOLLO_HOME
bash docker/scripts/install_docker.sh

sudo apt-get update
sudo apt-get install docker-ce
sudo groupadd docker
sudo usermod -aG docker $USER

echo "Please log in and out and run: bash apollo/docker/scripts/release_start.sh"
```

---

## 2

# Bag-file Simulation

## 2.1 Creating bag-files

To extract a topic from a bagfile use:

```
rostopic echo -b docs/demo_guide/demo_2.0.bag /apollo/perception/obstacles...  
... >> test.csv
```

To exclude the data for a certain module, i.e. remove the topics it is sending, use the following command:

```
rosbag filter demo_2.0.bag <../newfile>.bag 'topic != ''/apollo/<modulename>''
```

To start simulation using bagfiles use following command:

```
rosbag play -l <../filename>.bag
```

(The -l loops the bag-file, remove it if you want to run it just one time.)

Observe that the car follows existing GPS data and will not follow new routing commands or stop, it will run as it would when running the original bag-file.

To observe what messages are being sent over a topic run the original bag-file and use:

```
rostopic echo /apollo/<topicname>
```

The topicnames can be obtained either by running the following command:

```
rostopic list
```

Or you can go to "apollo/scripts/topics.txt"

If you want to see what topics are contained in a bag-file use:

```
rosbag info <filename>.bag
```

**Important: do not forget to turn on each module that is tested!**

# 3

## Complete system simulation

### 3.1 The three maps essential for simulation

There are three different maps that the system will use when simulating; `base_map`, `routing_map` and `sim_map`. These maps can be represented by `.txt`, `.xml` or `.bin` files. The most important one is `base_map`, which is also the most detailed of the three. `routing_map` contains the topology of the lanes in `base_map` and can be generated by the following two commands

```
$ dir_name=modules/map/data/demo # example map directory
$ ./scripts/generate_routing_topo_graph.sh --map_dir ${dir_name}
```

`sim_map` is a light weight version of `base_map` for Dreamview visualization. It has reduced data density for better runtime performance. It can be generated by the following two commands

```
$ dir\_name=modules/map/data/demo # example map directory
$ bazel-bin/modules/map/tools/sim_map_generator ...
... --map_dir=${dir_name} --output_dir=${dir_name}
```

Actually, `base_map` itself is generated from the HDmap, which is currently created and provided by companies like Mobileye. The creation process is very complex and involves driving test vehicles equipped with perceptive sensors in the area to be mapped, process the sensor data with deep learning algorithms, manually adjust roads, junctions etc and manually correct any observed inconsistencies. The `base_map.xml` file is based on a modified and expanded OpenDRIVE standard format, for which the format specification can be found in appendix

### 3.2 Creating a new map

Currently new maps need to be created by hand. There exists programs like OpenRoadEd (<https://openroad.ed.sourceforge.io>) for creating OpenDRIVE *standard* formatted maps graphically, but there does not seem to be a way of converting the standard format to the Apollo modified format yet.

A convenient way of creating a new map is modifying an already existing one, e.g. the `chalmers.xml` file originally created by the Perception group.

### 3.3 Changing maps

Use a different map, choose one option:

1. [Preferred] Change global flagfile: `*modules/common/data/global_flagfile.txt*`
2. Pass as flag, which only affects individual process:

```
bash
<binary> --map_dir=/path/to/your/map
```

3. Override in the module's flagfile, which generally located at:

```
modules/<module_name>/conf/<module_name>.conf
```

Obviously it also only affect single module.

```
txt
--flagfile=modules/common/data/global_flagfile.txt

# Overrides values from the global flagfile.
--map_dir=/path/to/your/map
```

### 3.4 Implement CAN for new vehicle

Git-info: [https://github.com/ApolloAuto/apollo/blob/master/docs/howto/how\\_to\\_add\\_a\\_new\\_vehicle.md](https://github.com/ApolloAuto/apollo/blob/master/docs/howto/how_to_add_a_new_vehicle.md)

Main work-folder: `modules/canbus/vehicle/`

The bare necessities for a car in Apollo is partly described in the "how to" of adding a vehicle. This only covers the initial steps of each task however. We haven't gotten it to register a new car doing only what is described in the how to. The major steps to be taken are (as described in the apollo git how to):

- In a new folder add the following:
  - A new vehicle controller (`twizy_vehicle_controller.h`)
  - A new message manager (`twizy_message_manager.h`)
  - A new vehicle factory (`twizy_vehicle_factory.h` and `.cc`)
- The car must be registered in the `vehicle_factory.cc`, `modules/canbus/vehicle/vehicle_factory.cc`
- The config file must be updated with the vehicles data, `modules/canbus/conf/canbus_conf.pb.txt`

By examining the already existing Lincoln you come to the conclusion that some extra steps are required, these mainly being:

- Adding "`#include`"'s to the correct other scripts which run the "car generation", eg. including the new car-scripts/classes into Apollo. Currently some/many scripts rely on the already existing

Lincoln code to generate the car virtually. To be able to do so with a new car, the new car must be included there as well.

- Adding the required information inside the new `vehicle_controller`, `message_manager`, `vehicle_factory` and config files. This information is in many ways the description of the car as a system/model, or at least what puts it all together.
- Inside the new vehicle folder there will need to be another folder called "proto". This folder in the Lincoln case contains several `.h/.cc` files describing everything from the wheelspeed, to the throttle, to the gps, to the fuel-level etc. etc.
- New protocols need to be added for the new vehicle. These follow the template of `ProtocolData (/modules/drivers/canbus/can_comm/protocol_data.h)`. As can be noted in the Lincoln case, these protocols can be quite substantial.

There is another group called Safety who succeeded in creating a new vehicle in Apollo, they said that if you have a `dbc`-file most of the code necessary was generated by itself.

### 3.5 Starting a new simulation

```
bash docker/scripts/dev_start.sh           # Initialize the dev-docker
bash docker/scripts/dev_into.sh           # Jump into the dev-docker
bash scripts/dreamview_sim_control.sh     # Start the simulation
bash scripts/dreamview_sim_control.sh stop # Stops the currently
      running dreamview simulation. Allowing for changes to be made.
```

A choice between `bootstrap` and `sim_control` will be necessary. It is recommend to use `sim_control` for simulation.

\*To kill all docker containers quickly, use "docker kill \$(docker ps -q)"

When this is run, go to `localhost:8888` in your web-browser, there press on "Module Controller" and turn on planing and routing. After that you can go to "Route editing", choose a start and final point then press "routing request". Now you will see the car move from the start point to the end point.

### 3.6 Adding obstacles to the simulation

To make the car react to objects they are added to the `obstacles` topic, to which the planing module listens to. To do this a function is added to the `sim_control.cc` file and in the function `SimControl::PublishLocalization(double lambda)` which runs every time the car position updates. The function that is added publishes to the `obstacles` topic.

To make the function able to publish to the `obstacle` topic the node it is written in have to be a publisher to the `obstacle` topic. This is done by updating the `adapter.conf` file so that the mode is `publish` or `duplex` (can both publish and subscribe): `config type: PERCEPTION_OBSTACLES mode: DUPLEX message_history_limit: 1`

If the function is not written in an already existing file the `cc` file and `.h` files should be added to the `BUILD` file in that folder that the code is located. Under `cc.library` the `.h`-file should be added under



the hdrs and the .cc-file should be added under srcs. And if there are any new dependencies those should be added under deps.

We have made a function called `updateSimWorld()` that is added to the last line in the `PublishLocalization` function. Because the function is in a other file the .h-file have to be included to the `sim_control.cc` file using the `#include` command.

`updateSimWorld()` is declaerd in a file called `sim_test`, what the code dose is that it creates a `perceptionObstacles`, which is a protobuf message. To the `perceptionObstacles` contains `perceptionObstacle`, time information and more. The `obstacle` contains information about its position, size, type, which ID it have and more. The `Obstacles` are then published to the `obstacles` topic using the `apollo::common::adapter::AdapterManager::PublishPerceptionObstacles(obstacles);` command.

## 3.7 Debugging

There are output files created under `apollo/data/log` that can contain good information, they can be written to using a command like:

```
AINFO << "The canbus conf file is loaded: "
```

And in the `apollo/data/core` crash files are created, it is good to sometimes remove those because they take up a lot of space.

## 4

# CAN

After the CAN-bus is up and running, which is indicated by a green-light by the CAN-bus in dreamview. The CAN-module have to be configured so that it can send actual messages over the bus. This has mainly been done by the safety grope of the bigger project.

But Apollo can auto generate some code if the ID's of the CAN-bus are known and what are sent over them. Then in that generated code it tells you where to translate the control command to a message on the can-bus and where to write cod that converts from can messages to chassis messages.

We ran in to some trouble while trying to get the can-bus code to send our messages instead of zeroes. So all checks in `vehicle_controller.cc` in the function `VehicleController::Update` at line 104 to 122 where removed which made our commands go through even if something was wrong. To compensate for the lack of checks the safety group added some functions to make the car safer. Safetys code where added in to the code which translates the code from control messages to CAN commands, what the code does is that it inactivates the control module if drive is not active or if the break-pedal is pressed.

Be aware that the can-bus sends the control messages no matter what, which means that if a simulation runs and the control and can module are activated the car will try to move blindly accordingly to how the car moves in the simulation.

Normally Apollo sends the speed as a throttle percentage and it uses some PID-regulators to convert the wanted speed into a throttle percentage. This have been modified because the motor-control group wanted the reference speed directly, the speed is then scaled so that 5 m/s is represented by 255 on the can-bus.

## 5

# Add an Adapter

To integrate the safety module in to Apollo the control module had to publish messages onto a topic that the can-bus module dose not listen to so that safety could stop the message to reach the can-bus.

This was done by creating a new adapter called ControlCommand2 which publishes on the topic called /apollo/control2. On that topic the control command are published which safety forwards to the old topic called /apollo/control if everything is as it should.

The new adapter was created by adding some code into the adapter module:

After line 272 in adapter\_manager.h the following line was added

```
REGISTER\_ADAPTER(ControlCommand2);
```

After line 101 in message\_adapters.h the following line was added

```
using ControlCommand2Adapter = Adapter<control::ControlCommand>;
```

After line 93 in adapter\_manager.cc the following line was added

```
EnableControlCommand2(FLAGS\_control\_command\_topic2, config);
```

After line 33 in adapter\_gflags.h the following line was added

```
DECLARE\_string(control\_command\_topic2);
```

After line 37 in adapter\_gflags.cc the following line was added

```
DEFINE\_string(control\_command\_topic2, "/apollo/safety", "control  
command2 topic name");
```

After line 46 in adapter\_config.proto the following line was added

```
CONTROL\_COMMAND2 = 40;
```

(MIGHT NOT BE NECESSARY, because it was not used in adapter\_manager.cc as a separate case)

And in the module control in the file control.cc at line 315

```
AdapterManager::PublishControlCommand(*control\_command);
```

was changed to

```
AdapterManager::PublishControlCommand2(*control\_command);
```