Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Bjørn André Aaslund

A Structured Approach to Autonomous Driving in Simulated Environments

Master's thesis in Computer Science
Supervisor: Rudolf Mester
June 2022

NTNU
Norwegian University of
Science and Technology

Bjørn André Aaslund

A Structured Approach to Autonomous Driving in Simulated Environments

Master's thesis in Computer Science
Supervisor: Rudolf Mester
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

Autonomous vehicles have made significant progress and gained much attention in the last years. This thesis studies a new approach for simulated autonomous driving with a focus on safety. There are currently two different families of approaches to autonomous vehicles; end-to-end approaches and structured approaches. A structured approach is selected for this thesis because each module can be tested and certified. These are crucial for the system's safety and essential aspects for the prominent stakeholders.

The simulation environment CARLA is selected as the simulation environment for this system. CARLA offers a wide variety of sensors. A semantic camera is one of these sensors, and it is used to create a bird´s eye view sensor that is used in this thesis. CARLA operates with many semantic classes that are not relevant for driving. Therefore, the semantic classes are mapped into four new semantic classes: road, moving object, not driveable object, and don't care. This modified bird´s eye view is used to create a compact but sufficient intermediate representation of the environment.

The center of the ego vehicle in the modified bird's eye view sensor sends out simulated rays for equally distributed angles around the vehicle's center, which end when they hit a moving object or a not driveable object. These simulated rays are used to calculate distances between the outer surface of the vehicle and other moving objects and not driveable objects. This intermediate environment representation is further extended by adding the semantic class and the relative velocity of the hit points. This intermediate environment representation does not contain information about the temporary goal pose of the vehicle. To include this information, a potential field containing information about the temporary goal pose of the ego vehicle is included. The potential field is also created from the modified bird´s eye view.

Planning requires the possibility of predicting how the autonomous vehicle will move for some control action signals. Nonholonomic constraints restrict the motion of vehicles, and therefore it is crucial to design a motion model that predicts feasible short-term plans that satisfies these restrictions. The design of the physical vehicle motion model is done by first performing system identification of the drive train and the steering input to the inverse radius of the circular path that the vehicle is following. The complete physical vehicle motion model makes it possible to create feasible short-term plans based on control action sequences. The control action sequence is a time segment of the steering and throttle signals over a limited time. These control action sequences can be compactly described by a small number of parameters, called *local plan parameters* in this thesis. In principle, such local plan parameters can be randomly chosen, tested for feasibility (collisions need to be avoided), and ranked in terms of suitable quality measures. However, the present work aims to train an agent to generate a small set of suitable local plan parameters based on the environment description. For this purpose, the system uses imitation learning and reinforcement learning. The best short-term plan is executed by applying lateral and longitudinal control or to simply execute the underlying control action sequence. The selected short-term plan is executed for a short time period before the autonomous vehicle creates a new short-term plan.

This results in an autonomous vehicle that is able to safely drive through a testing route with different obstacles. It is easy to test and certify each component of the complete system. The results also show that the autonomous vehicle drives on paths similar to clothoids in the curves.

Sammendrag

Autonome kjøretøy har gjort betydelig fremgang og fått mye oppmerksomhet de siste årene. Denne avhandlingen undersøker en ny tilnærming til simulert autonom kjøring med fokus på sikkerhet. Det er for tiden to forskjellige grupper av tilnærminger til autonome kjøretøy; ende-til-ende tilnærminger og strukturerte tilnærminger. En strukturert tilnærming er valgt i denne avhandlingen fordi hver modul kan testes og sertifiseres. Dette er avgjørende for systemets sikkerhet og vesentlige aspekter for de største interessentene.

Simuleringsmiljøet CARLA er valgt som simuleringsmiljø for dette systemet. CARLA tilbyr et bredt utvalg av sensorer. Semantiske kamera er en av disse sensorene, og det blir bruket til å lage en fugleperspektivsensor som brukes i denne avhandlingen. CARLA opererer med mange semantiske klasser som ikke er relevante for kjøring. Derfor er de semantiske klassene inndelt i fire nye semantiske klasser: vei, objekt i bevegelse, ikke kjørbart objekt og uinteressant. Det modifiserte fugleperspektivet brukes til å lage et kompakt, men tilstrekkelig representasjon av miljøet.

Sentrum av ego-kjøretøyet i den modifiserte fugleperspektivsensoren sender ut simulerte stråler for likt fordelte vinkler rundt kjøretøyets senter, som slutter når de treffer et objekt i bevegelse eller et objekt som ikke kan kjøres. Disse simulerte strålene brukes til å beregne avstander mellom den ytre overflaten av kjøretøyet og andre bevegelige objekter og ikke kjørbare objekter. Denne representasjonen av mellommiljøet utvides ytterligere ved å legge til den semantiske klassen og den relative hastigheten til treffpunktene. Denne representasjonen av mellommiljøet inneholder ikke informasjon om kjøretøyets midlertidige målposisjon. For å inkludere denne informasjonen, inkluderes et potensialfelt som inneholder informasjon om den midlertidige målposisjonen til ego-kjøretøyet. Potensialfeltet skapes også fra det modifiserte fugleperspektivet.

Planlegging krever muligheten til å forutsi hvordan det autonome kjøretøyet vil bevege seg gitt noe informasjon om miljøet rundt kjøretøyet. Ikke-holonomiske begrensninger (*eng: nonholonomic constraints*) begrenser bevegelsen til kjøretøy, og derfor er det avgjørende å designe en bevegelsesmodell som forutsier mulige veier som tilfredsstiller disse begrensningene. Utformingen av bevegelsesmodellen gjøres ved først å utføre systemidentifikasjon av drivverket og styreinngangen til den omvendte radiusen til den sirkulære banen som kjøretøyet følger. Den komplette bevegelsesmodellen gjør det mulig å lage gjennomførbare planer basert på kontrollhandlingssekvenser (*eng: control action sequences*). Kontrollhandlingssekvensen er et tidssegment av styrings- og gasssignaler over en begrenset tid. Disse kontrollhandlingssekvensene kan beskrives kompakt med et lite antall parametere, i denne avhandlingen blir dette kalt lokale planparametre (*eng: local plan parameters*) i denne avhandlingen. I prinsippet kan slike lokale planparametre velges tilfeldig, testes for gjennomførbarhet (kollisjoner må unngås), og rangeres i forhold til egnede kvalitetstrekk. Det nåværende arbeidet trener en agent til å generere egnede lokale planparametre basert på miljøbeskrivelsen. Til dette formålet bruker systemet imitasjonslæring (*eng: imitasjon learning*) og forsterkende læring (*reinforcement learning*). Den beste kortsiktige planen utføres ved å bruke lateral og langsgående kontroll eller ved å bare utføre den underliggende kontrollhandlingssekvensen. Den valgte kortsiktige planen utføres i en kort periode før det autonome kjøretøyet oppretter en ny kortsiktig plan.

Dette resulterer i et autonomt kjøretøy som trygt kan kjøre gjennom en testrute med forskjellige hindringer. Det er enkelt å teste og sertifisere hver komponent i hele systemet. Resultatene viser også at det autonome kjøretøyet kjører på stier som ligner clothoider (*eng: clothoids*) i kurvene.

Preface

This is a master's thesis in Computer Science, written in the spring of 2022 for the course TDT4900 at the Norwegian University of Science and Technology (NTNU).

In researching and developing a structured approach to autonomous driving, I have been able get a deep understanding of all the modules that is required for an autonomous vehicle. It has been very interesting to come up with ideas for all these modules, implement the ideas, and connect them together. I am grateful for the opportunity to work on this topic, and I hope that the reader will appreciate the synergy of these modules as much as I have done.

I want to thank my supervisor, Rudolf Mester. His clever ideas and guidance throughout the entire process has been extremely valuable and inspirational.

Trondheim, June 20th, 2022

Bjørn André Aaslund

Contents

List of Figures

List of Tables

1. Introduction

This chapter introduces the thesis by introducing and motivating the topic in section 1.1. section 1.2 describes the objectives of the thesis. Finally, section 1.3 briefly describes the outline of the thesis.

1.1. Introduction and motivation for autonomous vehicles

The human society is approaching a new paradigm in human travel, and *autonomous vehicles (AV)* are at the center of this evolution. Cars have evolved a lot during the last decades from containing no mechanisms for supporting the driver, to offering GPS-based navigation, emergency braking systems, obstacle alert systems, and adaptive cruise control to only mention some of the technologies new cars offer. Cars have advanced from completely mechanical devices to complex devices where diverse digital components accompany the mechanical main structure. The rapid growth and innovative developments are due to the many benefits they create for society. Completely autonomous cars will create new benefits for society regarding safety, independence, economy, and productivity to mention some.

National Highway Traffic Safety Administration (NTHSA) is an agency of the U.S. federal government, and reports that around 95% of all car accidents in the USA are related to human errors[1]. Thus by removing the human driver, the number of accidents can be decreased drastically. Autonomous driving has the potential to reduce risky behavior by drivers. Maybe the most significant impact might be that this approach gets rid of drivers who are tired or influenced by drugs.

By removing the need for a human driver, the cars no longer need to follow traditional design rules. People with disabilities that refuse them to drive traditional cars, can transport themselves independently with an autonomous vehicle. Even children could transport themselves by autonomous vehicles, and then they do not rely on another person driving them.

Autonomous driving could also create economic benefits. With fewer accidents, there will be less need for vehicle repair, which will also affect the medical costs. Both of these economic benefits will also be correlated with the safety aspect, causing fewer accidents. Fully autonomous driving could even improve the productivity of society. Today people use a considerable amount of their time driving, which the drivers could use doing other tasks. If the additional time is used to create value, this benefit might also influence economic growth.

Altogether the benefits of fully automated cars are enormous, and they will primarily have a positive impact on the society[2]. Autonomous driving is a research area that has received much attention. Many big companies are investing heavily in this technology. For the most prominent stakeholders, it is vital to be able to test and certify the autonomous cars. Since the liability requirements are so strict for the manufacturers, it is in the their best interest to test and certify the autonomous vehicle. Therefore, a structured approach to autonomous driving is developed in this thesis. A structured

[1] 2016 Fatal Motor Vehicle Crashes: Overview, `https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812456`

[2] The technology might be abused for military purposes.

approach to autonomous driving offer the opportunity to understand and verity each component of the system, which end-to-end approaches to autonomous driving currently do not. Therefore, this thesis aims to use a structured approach to design an autonomous vehicle for simulated environments that is based on a sparse but sufficient environment representation, which is used to create safe and feasible short-term plans.

1.2. Research objectives in this thesis

The overall goal of this thesis is to research and design a structured approach to autonomous driving in a simulated environment. The system is centered around safety, and therefore the design decision are made thereafter. The design of each component in the structured approach to autonomous driving is also done with the biggest stakeholders in mind. The goal is that the autonomous vehicle could be tested and certified. This will be done by designing a novel approach to create feasible short-term plans based on control action sequences and a physical vehicle motion model. Thus, the following research objectives have been formulated:

1. Research a sparse but sufficient intermediate environment representation.
2. Research the ability of creating a physical vehicle motion model for a simulation environment.
3. Research the ability of creating a physical vehicle motion model for a simulation environment.
4. Research the possibility of creating short-term plans that comply with nonholonomic constraints based on sequences of control actions.
5. Research the possibility of representing the control action sequences with few parameters that can be learned.

1.3. Thesis outline

This thesis starts by explaining two different approaches to autonomous driving in chapter 2, namely end-to-end and structured approaches. An overview of each approach is given, and the reason for selecting a structured approach for the autonomous vehicle is explained. Thereafter, the state of the art in autonomous driving and mobile robots is summarized in chapter 3. This chapter focuses on topics which are related to the design of our autonomous vehicle. CARLA is selected as the simulation environment for this thesis, and the relevant components from CARLA are explained in chapter 4. chapter 2 to chapter 4 contain background information necessary for this thesis. Next, an overview of the complete design of the autonomous vehicle is given in chapter 5. All the components of the system, and how they are connected are briefly explained. In chapter 6 we explain how the sensors from CARLA works, and how they are preprocessed to the sensor that is used by the autonomous vehicle in this thesis. The created sensor is the main building block for creating the intermediate environment representation that is explained in chapter 7. In chapter 8, the physical motion model of the vehicle is designed. The physical motion model and the intermediate environment representation is used to create short-term plans for the autonomous vehicle. This process is explained in chapter 9. The short-term plans are executed by a component explained in chapter 10. The methodology contains all the chapters from chapter 5 to chapter 10. The results from the final design of the autonomous vehicle is presented in chapter 11, and they will be discussed in chapter 12. The thesis will be concluded and further work will be presented in chapter 13.

2. End-to-end and Structured Approaches to Autonomous Driving

One way to divide AV systems into two groups is to separate end-to-end approaches from modular approaches. An end-to-end approach attempts to achieve autonomous driving using a single, comprehensive software component, while a modular approach contains several building blocks that together form the system. An overview of the two approaches is shown in Figure 2.1. An end-to-end approach could have an agent based on imitation learning or reinforcement learning, which takes sensor data as input and outputs actions. A modular approach divides the task of creating actions into several subtasks. These subtasks could be low-level perception, scene parsing, path planning, and vehicle control. By solving each of these subtasks, the goal is that the whole pipeline works as an AV. Each module in the modular approach may be partly, completely or not at all implemented by *machine learning (ML)*, and such a structured approach could also be trained end-to-end or whatever is possible for ML systems. The difference between the two approaches is whether they break down the task into different modules.

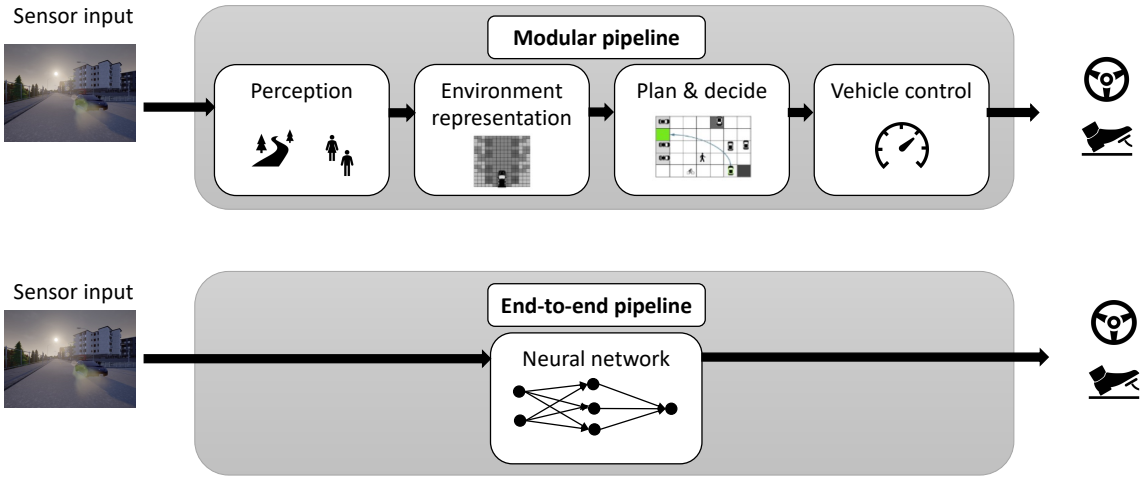


Figure 2.1.: Overview of the difference between an end-to-end approach and a structured approach to autonomous vehicles. The structured approach contain several components that required careful designing.

These two approaches have both advantages and disadvantages. End-to-end learning is simpler to implement in the sense that it requires less domain knowledge. It only requires one module that maps inputs to outputs, but this module might be very complex. These architectures need a lot

of data and require a long training time. The reasons for this is that the input space is large and complex, and driving is a hard task to learn. The ample input space makes it hard for the agent to generalize well. For example, if the sensor input is a RGB front-facing camera, then the agent needs to learn to stop behind other cars no matter what color they have. When the agent is designed end-to-end, it also removes the explainability of the agent. The agent creates its own rules, and it is hard to understand them for the designers. Explainability is a crucial aspect of safety-critical systems.

The added computational cost might not be a significant disadvantage of using an end-to-end approach for the most prominent companies. The biggest stakeholders like Google and Tesla have a lot of computing power and do not see this as the bottleneck for the AV. The critical aspect for the companies is the possibility of producing AVs where they can test each module. Since the liability requirements are so strict for the manufacturers, it is in the their best interest to test and certify each module. If an accident occurs, the manufacturer wants to know which part of the complete system caused the event, and they want to fix this problem without changing the rest of the system. However, this will make the complete system verification void, which require a new review.

The verification of each module and the complete system creates the need for each module to be explainable. It needs to be possible to understand what causes an error. Using raw sensor data, the agent gets noisy input with a lot of information that is unnecessary for driving. Thus, it is easier to explain the AV's actions if all the input data is relevant to the driving decision. One drawback of this is that one might remove relevant information without noticing it. It will further increase the explainability if each module is supposed to solve a specific task. Then it is possible to assess the decisions that each module makes, and the modules can also be tested and certified.

End-to-end approaches have shown outstanding performance in games like chess and go [Silver et al. 2016]. In these settings, the agents have learned a deeper understanding of the game by using end-to-end approaches. For AVs, the goal is not to develop radical new ideas but to create a robust, safe, and explainable system. A modular approach requires more expert decisions and knowledge, making the system harder to design. The different modules also often need to be trained independently, and it is harder to optimize the whole pipeline than for end-to-end learning. The advantages are that the AV will become more explainable, generalize better, and require less data than the end-to-end learning approach.

2.1. End-to-end approaches for AVs

There exist currently two main groups of end-to-end approaches for AVs, which is based on imitation learning and reinforcement learning. There have also been proposed combinations of these techniques to overcome the limitations of each of them.

2.1.1. Imitation learning for end-to-end autonomous driving

Imitation learning is a supervised learning technique that learns from expert decisions. The goal is to learn a mapping from input data which describe the current situation of the environment to the expert decision, thus it is required to collect data from an expert who explores the environment where the agent is going to learn to drive. For AVs, the input data could be a collection of sensor data or preprocessed sensor data, and the expert data could be a human driving in a city or an autopilot in a simulated environment. Given an input data x_i, the model parameterized with θ predicts an action $\hat{a}_i = F(x_i; \theta)$. The action contains information about how to turn the steering wheel and how much to push the gas pedal. The objective is to optimize a set of parameters θ, such

that the difference between the estimated action, $\hat{a}_i$, and the expert's action, a_i, is minimized. This can be expressed by Equation 2.1.

$$\min_{\theta} \sum_{i} (L(F(x_i; \theta), a_i), \tag{2.1}$$

where L is a loss function. This way of learning assumes that the expert actions only depend on the observations that the expert do. It is therefore important to give the learning agent all the information that the expert uses to make a decision. In intersections the expert decides where to turn based on the goal of arriving at a specific location. This is information that the learning agent also needs to take into consideration to be able to make the correct turn. If a goal location is set by the beginning of driving, a high-level planning algorithm can be used to find the best route. The navigational instructions given to the agent can either be the decision in the next intersection, or a set of the next way points.

Imitation learning has showed promising results for AV control the last couple of years. Some of the state of the art agents in simulated environments take advantage of this approach, for example Prakash, Chitta, and Geiger (2021). Although the promising results, there exists several limitations with this way of learning. In imitation learning the agent do not perform any exploration to learn what not to do in rare situations, or how to get back on track after a wrong decision. The agent only learns how to imitate an expert, which only performs a small subset of the possible actions. Imitation learning is also a data hungry approach, and it requires a lot of expert data to perform well.

2.1.2. Reinforcement learning for end-to-end autonomous driving

Reinforcement learning is a group of learning based algorithms where an agent performs actions in an environment after observing an environment state[1]. The agent gets feedback in terms of rewards which reflect how good the action is. Most research in reinforcement learning has primarily focused on solving video games and robotics locomotion problems. There has been relatively limited research on the use of deep reinforcement learning for end-to-end autonomous driving [Kiran et al. 2021], perhaps since it is hard to safely train a reinforcement agent in the real world since such a reinforcement learning agent needs to explore the environment to learn. There is, however, recent notable work done by Kendall et al. (2019).

The main areas of designing a reinforcement learning based agent for end-to-end autonomous driving is to select an appropriate environment, defining the environment state, selecting an appropriate reinforcement learning algorithm, and designing a reward function. The reward function is often a combination of positive rewards for desirable driving behaviours, and negative rewards for undesirable driving behaviours as collision and driving in the wrong lane. The reward function is often a weighting of different rewards, and it might be challenging to select the appropriate weights. Some possible rewards to use in AVs are closeness to the speed limit, distance from the middle of the desired lane, difference between orientation of the vehicle and the desired lane, and measures for comfort. Kiran et al. 2021 presents an extensive survey on deep reinforcement learning for autonomous driving.

However, there are some significant drawbacks when using reinforcement learning that impedes its applicability to control tasks, in particular the lack of guarantees for safe operation and the capability to specify constraints. Some of these limitations can be addressed by combining reinforcement

[1]The input to reinforcement learning algorithms is usually defined as a state. In this thesis, I define the state input as an environment state to not cause confusion with the vehicle states that are used in the vehicle motion models

learning with reliable control approaches [Bøhn et al. 2021]. These drawbacks can be handled by using reinforcement learning as a component in a structured approach to autonomous driving instead of using it in an end-to-end approach.

2.2. Structured approaches to AVs

The development of an AV is a complex task that can be divided into modules which can be solved on their own. This requires more domain knowledge than the end-to-end approaches, but it offers explainability and handles some of the current limitations with the structured approaches. It is also possible to test and verify different modules. The AV task is divided into perception, environment representation, planning and deciding, and control.

2.2.1. Perception

The first module is about how the AV senses the environment. The perception is the information the AV has available to plan and make decisions. Different sensors are required for obtaining different observations, which can be used in different parts of the system. For example, a camera can be used to detect traffic signs, while a radar can be used to calculate the distance to the vehicle in front of the ego vehicle. The AV needs different sensors to perceive the environment far away from the vehicle and close to the vehicle. Redundant sources also increase confidence in the detection and are important for robust sensing which improves safety. LiDAR, camera, radar, and ultra-sound are sensors that are commonly used for autonomous driving. The different sensors that are used for the perception of the environment offer information on different levels of abstractions, and the sensor measurements need to be preprocessed to be meaningful for an AV. The different sensor measurements are used in different parts of the environment representation.

2.2.2. Environment representation

The sensors include a lot of information that is not necessary for driving a car safely, and the measurements are often noisy. Environment representation and localization are used to tackle these problems. An environment representation for an AV should include all the necessary information for an AV to drive. Environment representations are about extracting useful information from sensor measurements. Semantic segmentation and object detection are two procedures to separate different objects, and creates a better understanding of the environment. Another environment representation that is currently popular by designers of AVs is the so-called *bird´s eye view (BEV)* representation[2]. This is a top down view of the environment around the ego vehicle, and might include information about the drivable area, the position of other relevant objects, and the desired lane. Although, the BEV does not contain any dynamic information, and hence it needs to be accompanied by additional velocity information to create a sufficient environment representation for autonomous driving. A sufficient representation of the surrounding environment is required to plan where to drive.

2.2.3. Plan and decide

Planning is often divided into navigation and short-term planning. The navigation is a high-level plan about how to drive from the start position to the goal. This is similar to the navigational

[2]Tesla currently uses a high-fidelity BEV representation in their autonomous vehicle design: `https://www.tesla.com/AI`

information drivers currently get from a GPS-based routing device. One approach to obtaining such a high-level plan is to treat the road network as a weighted graph. The shortest path is found by a shortest path algorithm as A^*. Navigational instructions are generated from the resulting shortest path. The short-term planning is about generating a trajectory over a short time horizon that is safe, efficient, comfortable, and approaches the global goal location. This plan incorporates several decisions based on the perception. The decisions can for example be to stop in front of a red light or drive around an obstacle. These types of decisions are often handled with finite state machines.

2.2.4. Control

The last module executes the short-term plans. This is done by creating actions that make the vehicle follow the short-term plan. The control task can either be divided into lateral and longitudinal control, or be handled in one unit. Popular approaches are classical methods of closed loop control and optimal control. The classical methods of closed loop control execute actions which compensate for the deviations between the plan and the observed motion. A popular control design is to use a PID (proportional-integral-derivative) controller for the longitudinal control, and a geometric controller as a pure pursuit controller or a Stanley controller for the lateral control. The optimal controllers select actions for a short time horizon such that a cost function which depends on the actions is minimized. The optimization problem can also be extended by constraints, and *model predictive control (MPC)* is a popular optimal controller that contains constraints. The goal of MPC methods is to track a specified path while stabilizing the behaviour of the vehicle [Paden et al. 2016].

3. State of the Art

Bojarski et al. (2016) create an AV that can follow a lane based on a *convolutional neural network (CNN)*. They explain that this approach optimizes the whole pipeline at once but needs more work to improve robustness. One way to improve robustness is to create a sufficient environment representation that removes unnecessary information. Bird's Eye View (BEV) is a map where the point of view is above the vehicle. BEV is assumed to be a sufficient environment representation since autonomous driving can be simplified to a system that operates in the plane. Roddick and Cipolla (2020) uses 8 RGB cameras to create a semantic BEV. Philion and Fidler (2020) takes this further by applying motion planning in the outputted BEV. Toromanoff, Wirbel, and Moutarde (2020) take another approach by designing different representations they assume to be necessary for autonomous driving. They train an autoencoder to extract information about the semantics, traffic lights, intersections, and lane positions. The encoder part is used to represent the environment, and actions are selected based on this information. A similar approach to extract features is used by Chitta, Prakash, and Geiger (2021). This approach uses attention maps to create efficient reasoning about the environment's semantic, spatial, and temporal structure. A sparser environment representation is used by Klose and Mester (2019), which uses distances from the vehicle to the curb of the road. They define this representation as a Circogram, which will be extended in this thesis.

In the last years, AVs have been closely related to learning. Well designed simulation environments have made it easier to design and test different learning approaches for AVs that are dangerous to test in the real world. There have been many innovative approaches to use imitation learning for AVs in the years after the success by Bojarski et al. (2016). Chen et al. (2020) divide the imitation learning task into two steps. In the first step, they create an agent with access to the ground truth information. This agent then learns to drive by using all the available information, which they call learning by cheating. In the second step, this agent supervises another agent that only has access to image data. Chen et al. (2020) highlight that further work should combine this with reinforcement learning to exceed the capabilities of the expert. Codevilla et al. (2019) explain the limitations of imitation learning when scaling to a full specter of driving scenarios. Generalization issues are one of the limitations that are highlighted. The generalization issue can be overcome by increasing the dataset or introducing reinforcement learning. Liang et al. (2018) introduce an approach called *Controllable Imitative Reinforcement Learning (CIRL)*, which combines reinforcement learning with imitation learning to make learning more data efficient. Furthermore, they apply different specialized policies and reward functions for different situations. Liang et al. (2018) was the first approach to use reinforcement learning to learn a policy in a simulator and obtain a better performance than approaches based on imitation learning. Toromanoff, Wirbel, and Moutarde (2020) argues that there currently does not exist a reinforcement learning algorithm that can handle urban driving. Hence, some preprocessing is necessary to utilize reinforcement learning. Toromanoff, Wirbel, and Moutarde (2020) creates an autoencoder that extracts implicit affordances, which are used in the reinforcement learning setup. The previous papers use the learned policy in all driving situations, but Ohn-Bar et al. (2020) highlights that different driving behaviors are required in different situations. Therefore, they propose a mixture of policies that are used in different situations. The mixture of policies makes the AV perform well in diverse situations.

The learning algorithms presented in the previous paragraph have learned to output different vari-

ables. Liang et al. (2018) outputs the next action that should be used. This means that it outputs 3 values between -1 and 1, corresponding to steering, throttle, and brake. Chen, Koltun, and Krähenbühl (2021) discretize the actions and output a categorical distribution over the discrete actions. Only outputting the actions may lead to noisy behavior. To overcome noisy driving, Chen et al. (2020) predicts the next waypoints that the AV should drive through. Chitta, Prakash, and Geiger (2021) uses a similar procedure but includes samples from a uniformly spaced grid for robustness. One disadvantage of this approach is that the vehicle might not able to drive through the waypoints due to the nonholonomic constraints of the vehicle.

Optimal control is a well studied area and has great success in mobile robotics. *Model Predictive Control (MPC)* is a way to control a process while fulfilling constraints at the same time [Rawlings and Mayne 2009]. MPC has become popular due to its capacity to handle complex nonlinear dynamics while still satisfying output constraints. This has been a successful approach for planning and controlling when solving complex quadrotor problems. Falanga et al. (2018) creates a MPC based model that combines planning and perception of objects. Regardless of the success, MPC still encounters problems in some applications due to high computational costs, the need for an accurate mathematical model, and the sensitivity to design choices. The design choices that heavily influence the performance are the formulation of the cost function, hyperparameters, and the prediction horizon. These limit the MPC to generalize well for problems that need different design choices. Lately, there have been approaches that aim at utilizing the strength of MPC and learning to make models more flexible. Song and Scaramuzza (2020) propose to use a policy to output high-level decision variables for a MPC controller. They show that this approach overcomes some of the difficulties with MPC. This approach is extended and tested for agile drone flights by Song and Scaramuzza (2022).

This thesis aims to develop a robust model for the AV, which uses a sufficient environment representation to compute feasible short-term trajectories. Based on the recent improvement in combining MPC with learning, we want to utilize the well-known dynamics of vehicles and reduce the complexity of the learning problem.

4. Overview of the Relevant Components of the CARLA System

Realistic *digital simulation environments* are becoming one of the most important concepts to build safe AV technology. The use of a simulator that has realistic physics, the necessary sensors, and possibility to generate a wide range of driving scenarios is a key to build safe and robust AVs. The complexity of urban environments requires that models need to be tested in countless environments and traffic scenarios to be able to generalize well. This requirement causes the cost and development time to exponentially increase using the *physical real world environment*. For this reason a simulator is vital.

In 2017 there existed few open source simulators suitable for AV research. Furthermore, the limited ones were quite restrictive in terms of customization and control over the environment. Dosovitskiy et al. 2017 observed the need for an open source simulator specifically designed for training and benchmarking AV systems, and created CARLA (Car Learning to Act). CARLA has now become one of the most powerful and promising simulators for developing and testing AV technology.

CARLA has been created to support the development, training, and validation of AV systems. In addition to open-source code and protocols, CARLA provides urban layouts, buildings, and vehicles that were created for this purpose and can be used freely. The simulation platform supports flexible specification of sensor suites, environmental conditions, complete control of all static and dynamic actors, map generation, and more. The simulator is an open-source layer on top of Unreal Engine 4 and is designed as a server-client system. The server runs the simulation and renders the scene, while the client is an interface that allows interaction between the autonomous agent and the server. All the major components in CARLA that are used or modified are described below.

4.1. Disable environment objects in CARLA

All objects in a CARLA simulation is assigned an ID when the simulation starts. It is possible to disable an object from a CARLA simulation by ID. This feature is used to disable all the objects that can interfere with the custom sensor described in section 6.6. The sensor is based on a camera above the vehicle which works as a BEV. The objects that are disabled consists of poles and low hanging vegetation. These objects might cover the road on the custom BEV sensor, and are therefore disabled. The objects that are disabled are static and hence will not influence the simulation other than being an obstacle. The disabled obstacles are located on the sidewalks, where the our implemented autonomous vehicle is very unlikely to drive. Therefore, the disabling of these objects will be beneficial for the custom sensor, but not reduce the difficulty for our autonomous vehicle.

4.2. Adjustment of vehicle physics in CARLA

CARLA offers the possibility to adjust the physics and the mechanics of the vehicles in the simulation. We will not use an electric vehicle in this work since the recuperation effect will complicate

the physical motion model in chapter 8. The standard Ford Mustang is selected for the ego vehicle, but other vehicles could also be used. Automatic gearing is selected, and the gear switching time is set close to zero. A gear switching time set to zero is again an action to increase the simplicity of how the throttle impacts the velocity. The reduction of gear switching time will remove tiny drops in the velocity when automatic gear changes occur. When a constant throttle or bake is applied, these changes to the vehicle physics will result in a smooth rise or fall in the velocity.

4.3. Relevant measurements and sensors in CARLA

A precise description of the location and the movement of the vehicle is required to perform the planning described in chapter 9. Therefore, we require several types of measurements in the design of our autonomous vehicle. The location of the vehicle is described with global coordinates in meters, and the orientation in angles. The world coordinate frame is described in chapter 8. The orientation is measured between the global x-axis and the longitudinal axis of the vehicle. The movement of the vehicle is described by the velocity and the rotation around the centre of gravity. Carla provides the velocity in m/s and the rotation as the change in angles per second. All the measurements that the server provides in angles are transformed into radians. The server provides these measurements for all the actors in the simulation. The measurements and the sensor data are necessary for creating the intermediate environment representation presented in section 7.1. It is possible to retrieve all these measurements every time the server ticks.

In this thesis, only two sensors of the vast amount of sensors that CARLA provide is used. LiDARs, RGB cameras, and semantic cameras are used. CARLA offer a simple way to obtain the sensor measurements at every tick at the server. It is important to notice that CARLA uses the Unreal Engine coordinate system where positive x is forward, positive y is right, and positive z is up. All the sensors return the coordinate measurements in local space. How these sensors are implemented will be further explained in section 6.2 and section 6.4.

4.4. High-level planning provided by CARLA

A component which creates high-level plans between two locations is created by CARLA. This component is not included in the CARLA package, but comes as file with a implemented class which can be imported. The start position of the high-level plan is usually the *spawning position*, which is a location where a vehicle can be teleported to without causing a collision. This component and how it is used in the autonomous vehicle is thoroughly explained in subsection 9.2.1.

4.5. Traffic Manager for creating specific driving scenarios

The Traffic Manager is a component in CARLA which controls vehicles in autopilot mode in the simulation. This module is used to populated the simulation with other vehicles, and to set up scenarios where the ego vehicle need more training or testing. In this thesis, only the velocity, the preferred distance to the leading vehicle, and the starting location of the other vehicles are controlled. The Traffic Manager also includes other more complex functionality that is not used in this thesis.

5. Overview of the SafeRide System

This chapter presents the overall structure of the system SafeRide developed in this project. SafeRide is a system that implements a particular, structured approach to simulated autonomous driving. This choice of the system architecture, the decision to implement the system as a composition of individual functional modules, makes it possible to test and certify each component. SafeRide consists of several components, which will be presented and explained in this section. Figure 5.1 shows an high-level overview of all the components and their connections. The architecture consists of a simulator based on Unreal Engine, an agent connected to a vehicle motion model, and an interface between the simulator and the agent. The simulator simulates vehicles and provides start and end locations for the drive, the *dynamic vehicle state* (position, velocity, orientation, and change in orientation) described in chapter 8, and a *semantic BEV* described in section 6.5.

The start and end locations are only provided once before the autonomous vehicle starts to drive. The vehicle is teleported to a new start location if the vehicle collides or reaches the end location, SafeRide and CARLA are used in an episode-based manner. The duration between the vehicle starts to it reaches the end location or collides is called an *episode*. An episode-based manner is used for collecting training data, training the agent, and to test difficult situations several times. The start and end locations are drawn randomly from a predefined set of possible spawning positions before every episode starts.

The interface uses the start and end locations, the dynamic vehicle state, and the semantic BEV to create a *navigation command* described in subsection 9.2.1, and an intermediate environment representations. This data together with the dynamic vehicle state is forwarded to the agent. The agent uses this information to create steering and throttle commands which are sent to the simulator through the interface.

5.1. The interface to the CARLA simulator

The autonomous driving simulator CARLA already introduced in chapter 4, is used as the platform on which the SafeRide system is developed. Figure 5.2 shows the three building blocks of the interface to the CARLA simulator. The *high-level planner (HLP)* is a class implemented by CARLA which takes the start and end locations as inputs and creates a high-level plan between these two locations. This high-level plan only takes the road topology into account and creates a route proposal. The route consists of equally distributed points between the start and end locations, which are extended with the desired speed and the *navigation command* which is a command describing the maneuver (left, right, straight, follow lane) at the location. An A* search with distance heuristic creates the high-level plan. This component is further explained in section 9.2. The high-level plan is the only input to the *navigator (NAV)*. This component uses the high-level plan to *navigation instructions* similar to a GPS-based navigation system that vehicles currently use. A navigation instruction is created from the next point in the high-level plan where the navigation command changes, explained in section 9.3. The navigation instruction consists of the distance along the high-level plan to this point, the current navigation command, and the next navigation command. An interpretation of an example of a navigational instruction could be, follow lane for 200 meters, then take left.

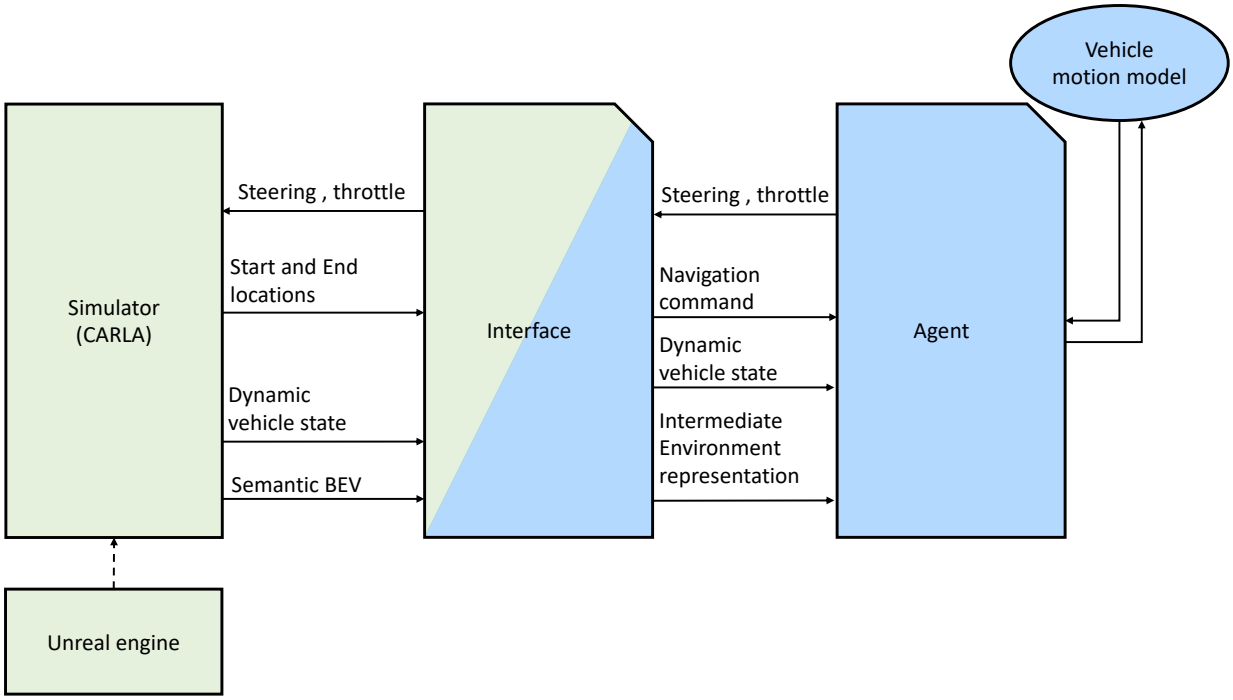


Figure 5.1.: High-level overview of the SafeRide system. The green boxes are provided software and implementations, and the blue boxes are implemented by me. The interface consists both provided software and implementations by me, and is therefore green and blue.

In order to allow autonomous driving, the simulator is used to provide measurements and sensor data, which the *state data preprocessor (SDPP)* preprocesses. The SDPP takes the dynamic vehicle state, the semantic BEV, and the high-level plan as inputs. The only transformation the SDPP does to the dynamic vehicle state is to change the orientation and change in orientation to radians, before it is forwarded to the agent. The SDPP also uses the provided input to create the bipartite intermediate environment representation. The first part of the intermediate environment representation include information of the distances, the relative velocity, and the labels of the surrounding objects. This intermediate environment representation is called *Circogram* and is described in . The second intermediate environment representation is a *potential field* which is a field of positive values that incorporate how far away a pixel is from the desired location in the semantic BEV. The high-level plan is used to find the attraction point in the semantic BEV, which is then used to create the potential field. The pixels that are not in the *free space* are assigned a high value. The potential field is furhter explained in .

5.2. The agent in the SafeRide system

Figure 5.3 illustrates the components in the agent and their connections. The *short-term planner (STP)* component obtains the navigation command, the dynamic vehicle state variables, the potential field, and the Circogram from respectively the HLP and the SDPP, and learns to create *local plan parameters*. A *short-term plan* is a feasible plan for the next couple of seconds which complies with the nonholonomic constraints of a vehicle; section 9.4 explains the details. This component learns to output safe local plan parameters from the provided data. A safe short-term plan is a plan that has a high *time to collision (TTC)* and keeps a certain distance to the other objects. The short-term plan differs from the high-level plan in the sense that the precise geometric structure and semantic labels, as well as the dynamics of the ego vehicle and surrounding objects are taken into

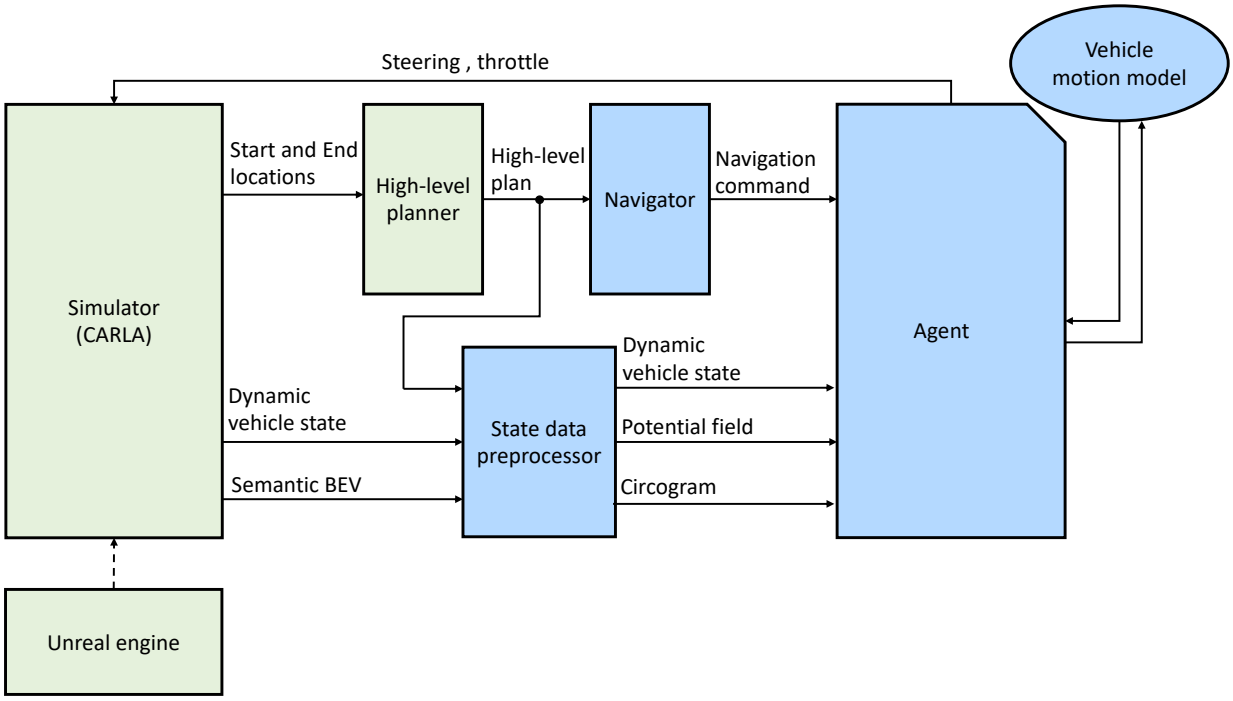


Figure 5.2.: Overview of the interface between the simulator and the agent. The interface consists of a high-level planner, a navigator, and a state data preprocessor.

consideration. A short-term plan is a trajectory over time, that is a sequence of dynamic vehicle states. In our approach, a short-term plan is represented by a small number of parameters (for details, see section 9.4).

The *dynamic action generator (DAG)* generates proposals for the actions the ego vehicle should perform during a short time period, which we call *control action sequences*. The goal of the STP is to create local plan parameters that can be used to generate several control action sequences by the DAG. The DAG uses the local plan parameters from the STP and a *motion model (MM)* to create feasible short-term plans. The *short-term plan selection (STPS)* starts by removing the short-term plans that lead to a collision (hard constraints), and selects the best of the remaining ones, based on risk, efficiency, and comfort. The *short-term plan executor (PE)* uses closed-loop control and the short-term plan to calculate control actions that are sent to the input channels of the simulated vehicle. The selected short-term plan is executed by a controller that compensates for the deviations between the plan (which is based on a non-perfect dynamic model) and the observed motion, or by executing the control actions in the control action sequence that is used to create the best short-term plan. The STP produces new local plan parameters after a given period, and the currently executed short-term plan is replaced by the updated one.

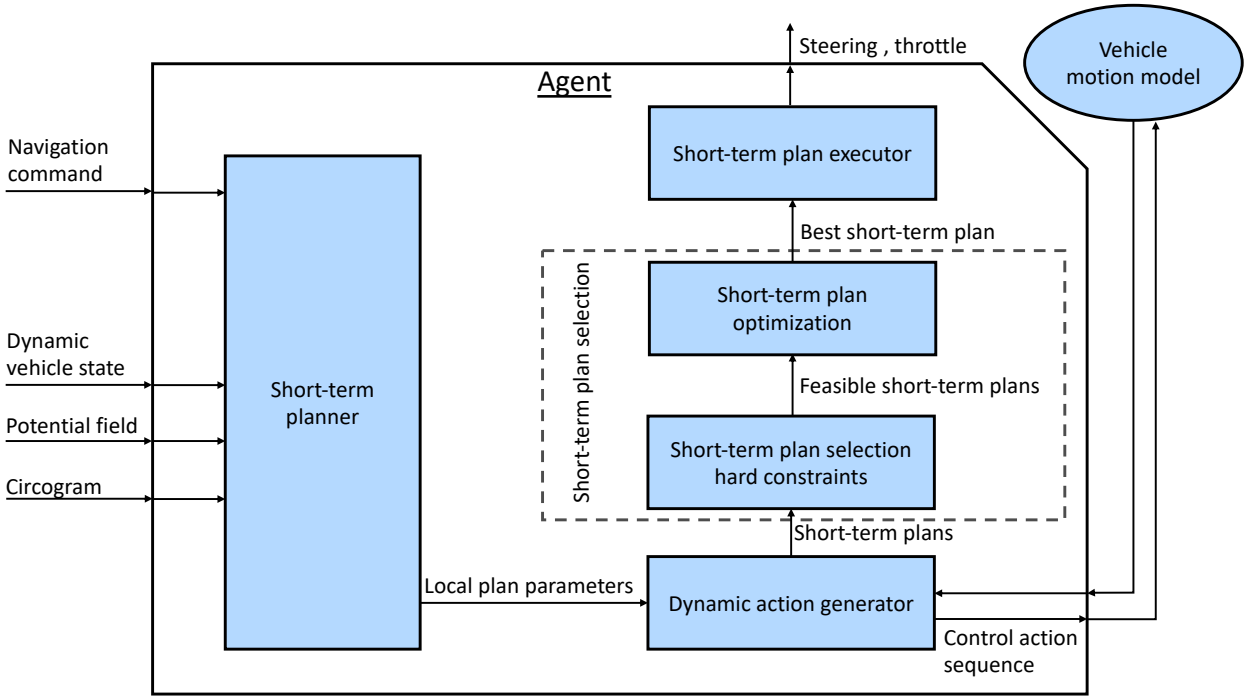


Figure 5.3.: Overview of the agent in the SafeRide system. The vehicle motion model is not a part of the agent, but is illustrated for completeness.

6. Sensor Data Processing in the SafeRide System

An autonomous vehicle needs the ability to detect the environment to act intelligently. Humans use their eyes and ears to see and hear what is happening around them. Humans can navigate their cars around obstacles and follow traffic rules by using this information.

Autonomous vehicles need some information about the environment to behave appropriately. Different sensors enable autonomous vehicles to perceive the environment in a similar way as humans, and some sensors give information to the autonomous vehicle that humans can not obtain. It is essential to select sensors that can be implemented in the real world and contain the necessary information to drive a car safely. This section examines LiDARs and cameras as sensors for autonomous vehicles.

6.1. LiDAR for autonomous vehicles

A LiDAR is a sensor with the ability to measure distances to other objects. This is done by sending out a rotating laser beam and measuring the time it takes for the reflected light to arrive back at the sensor, similar to a Radar system. The time can then be used to create an accurate but semi-sparse 3D plot of the surface of the objects around the sensor. It is a semi-sparse representation of the environment, as only a limited number of laser beams (Velodyne offers LiDARs with up to 128 laser beams) can be sent out, which gives a scanning of the environment in a special geometry. An example of a point cloud from a scanning is given in Figure 6.1 This means that it is possible to calculate the distance to the surrounding objects from the measurements. This distance is valuable information for a vehicle with the primary goal of not crashing with other objects.

The LiDAR sensor has some shortages when it comes to perceiving the surroundings. It cannot obtain information of traffic signs, the state of traffic lights, or road lane marking. This means that an autonomous vehicle also needs other sensors to follow the traffic rules.

6.2. LiDAR as implemented in CARLA

CARLA uses ray-casting to implement the LIDAR sensor, which was first introduced by Roth 1982. Simulated light rays are cast from the LiDAR sensor, and the paths are traversed. Ray-casting emissions detect mesh models in the Unreal Engine simulation, and generate the point cloud. The output raw data format is an array of 32-bit floats. Every four values are one point $p_i = [x_i, y_i, z_i, r_i]$, where x_i, y_i, z_i are coordinates and r_i is the intensity. The intensity is calculated from the length of the simulated light ray.

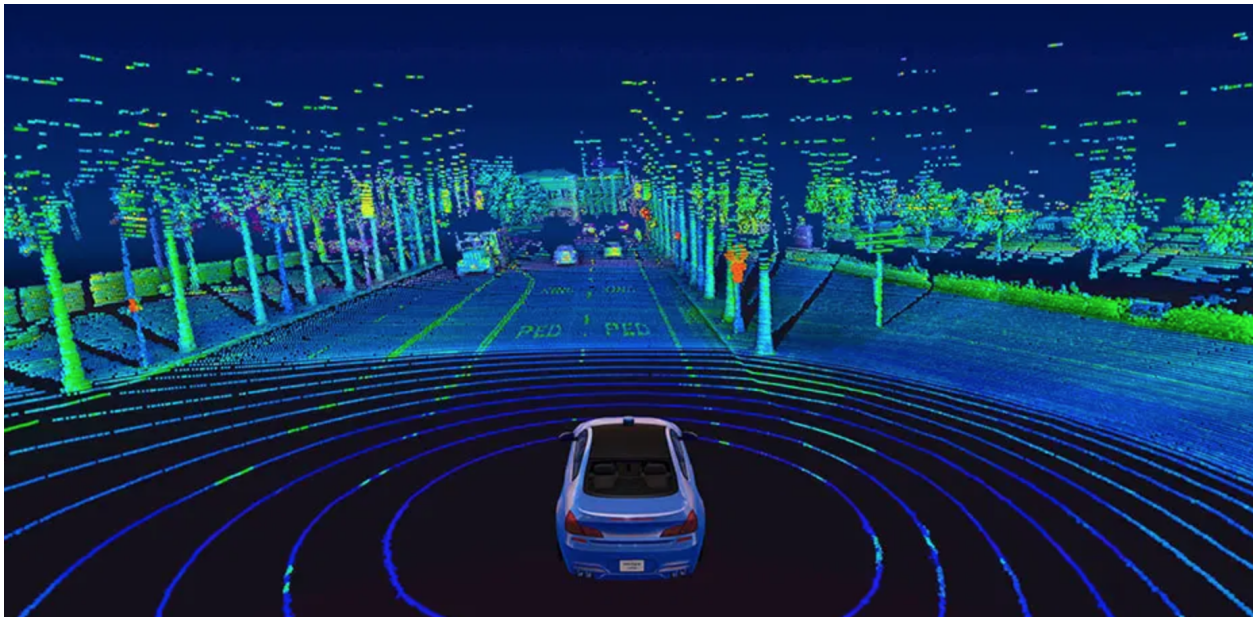


Figure 6.1.: A point cloud from a scanning of a LiDAR. Each circle around the car corresponds to a laser ray.

Source: https://velodynelidar.com/blog/guide-to-lidar-wavelengths/

6.3. Camera for autonomous vehicles

Cameras are the sensors that are most similar to how humans see the world. Hence this would be a natural choice of sensor. A camera is relatively inexpensive compared to the other sensors, but cameras require suitable software and affordable processing hardware to extract useful information. The software and hardware are needed to convert the image information into some representation that is useful for guiding a vehicle, for instance a semantic map (illustrated in Figure 6.2) or a point cloud. The usefulness of cameras is decided by the software that converts images into useful information for autonomous driving. By using the appropriate software, a camera can be used to detect both static and dynamic obstacles. These capabilities are beneficial for an autonomous vehicle and are in fact the only information needed for a human to drive. The detection of obstacles can give information on traffic signs, the state of traffic lights, road lane markings, and pedestrians.

This sensor is the only sensor presented in this section that has the capability of sensing changing driving conditions. A camera can capture icy parts on the road, and the driving behavior can be adjusted to the conditions. A challenge for cameras in Nordic environments is the possibility of snowy weather. An autonomous vehicle that has learned to follow the road lane marking may lose information needed for steering. This challenge can be overcome by including these edge cases in the training. The result might be that the autonomous vehicle learns to follow the wheel tracks of the other car

One camera can only detect what is happening in a particular part of the surroundings around the vehicle. Roddick and Cipolla 2020 proposed to use several cameras to incorporate the human possibility of changing the viewing direction by turning the head. As described in chapter 3 Tesla uses eight cameras to capture everything that is going on around the vehicle. One drawback of this solution is that many images do not contain useful information.

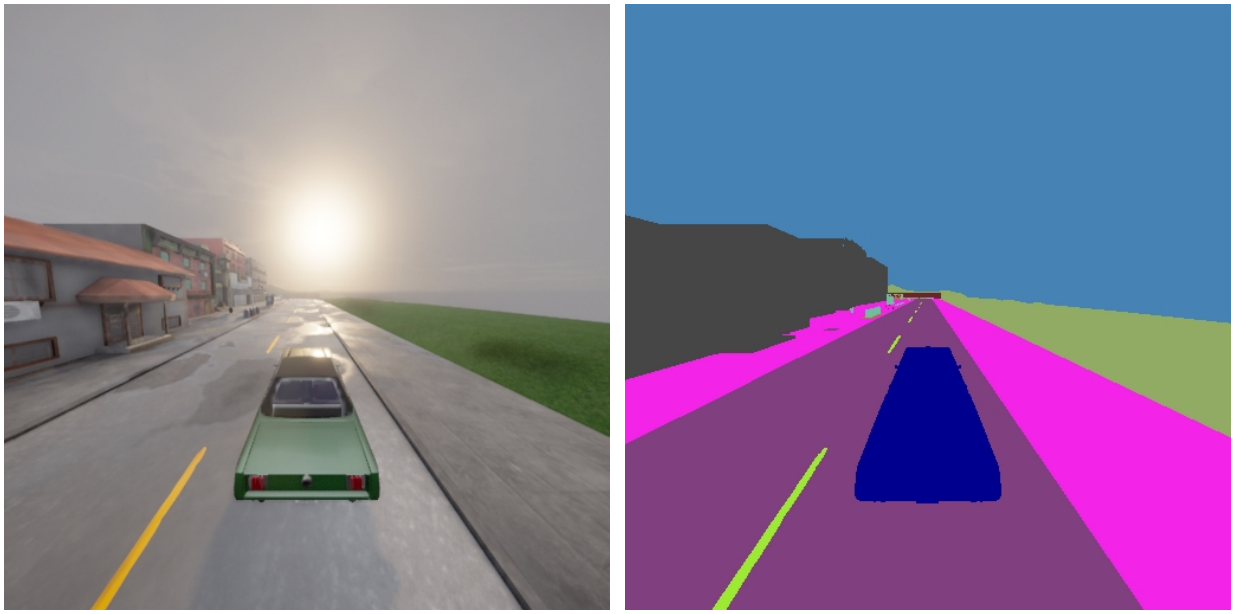


Figure 6.2.: Example of a conversion of a image to information that is relevant for guiding a vehicle. In this example the useful information are semantic classes. This conversion is performed in CARLA.

6.4. Cameras as implemented in CARLA

The cameras in CARLA is also implemented with ray casting in the same way as the LiDAR sensors. Simulated light rays are cast from the camera sensor, and the paths are traversed. Ray-casting emissions detect mesh models in the Unreal Engine simulation which are projected to a 2D plane. The cameras are initialized with field of view, height, and width.

6.5. Semantic sensors in CARLA

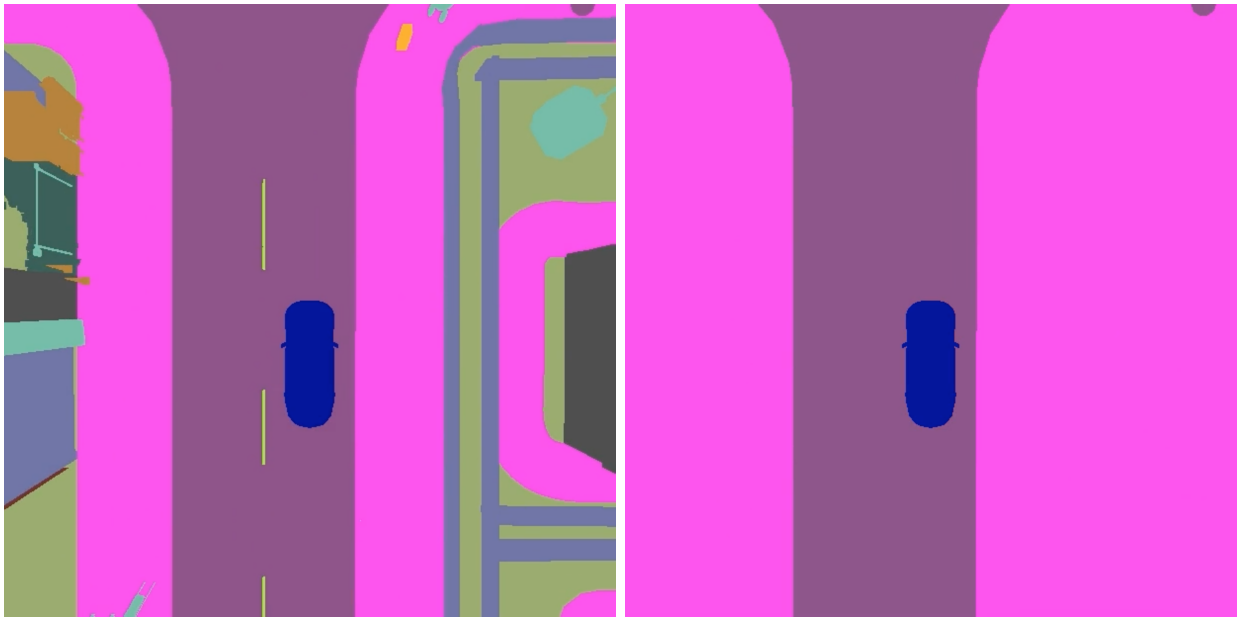
CARLA offers two different semantic sensors in their current version (version 0.9.13), namely semantic LiDAR and semantic camera. Both of these sensors are extensions of the original sensors described in section 6.2 and section 6.2. The semantic LiDAR is extended by the semantic class of the object that the simulated light ray hits. This means that each point in the point cloud contains five values. Each object in the simulation is assigned a semantic class when the simulation starts, and each semantic class corresponds to a color. The semantic camera in CARLA returns an image where the semantic class information is encoded in the red channel. The classes need to be mapped to the corresponding colors on the client side to view the semantic images. CARLA uses 23 different classes[1].

6.6. Recolored BEV sensor

Bird´s eye view (BEV) is an environment representation often used in autonomous driving. Tesla proposed a sophisticated way of generating BEV images from cameras in their latest AI-day presentation [2]. A BEV sensor does not exist in CARLA, but it is possible to create a ground truth BEV

[1] All the classes are listed here, `https://carla.readthedocs.io/en/latest/ref_sensors/#semantic-segmentation-camera`

[2] The complete presentation can be view here: `https://www.youtube.com/watch?v=j0z4FweCy4M`



(a) BEV with original semantic classes from CARLA. (b) BEV after recoloring to our own classes.

Figure 6.3.: Creation of custom recolored BEV sensor in CARLA.The pink color represent a not drivable object, the purple color represent the road, and the dark blue color represent a moving object.

sensor by placing a semantic camera above the ego vehicle and facing it downwards. The camera is placed in front of the vehicle since the information in front of the vehicle is more important than the information behind the vehicle. As explained in section 6.5, CARLA comes with many different semantic classes. All of these classes are not necessary information to drive safe, and hence these classes are reduced to only contain classes that are useful. In this thesis, the relevant classes are *road*, *moving object*, *not driveable object*, and *don't care*. Figure 6.3b illustrates the mapping from the semantic classes in CARLA to the relevant classes in this thesis. The *recolored BEV* is used as a sensor in the SafeRide system.

7. Intermediate Environment Representation

The raw sensor data is not immediately helpful in guiding an autonomous vehicle. Therefore, it is appropriate to preprocess the raw sensor data to an explicit intermediate environment representation where it is easier to learn how to drive. The hypothesis in this thesis is that it is easier for an autonomous vehicle to learn to drive safely and with more comfort for the passengers if an environment representation based on human understanding of physics, possible geometric structure, and traffic semantics is designed and made available to the learning agent. Figure 7.1 shows an example of such a transformation, where an algorithm transforms image data into *stixels* representing objects that can cause a collision with the ego vehicle. Another benefit of an intermediate environment representation is that it improves generalization. Figure 7.2 illustrates how an intermediate environment representation reduces the input space and improves generalization. The explicit intermediate environment represents the essential semantic, geometric, and kinematic situation. Two different intermediate environment representations are used in this thesis. The Circogram representation will be explained in section 7.1, and the potential field representation will be explained in section 7.2.

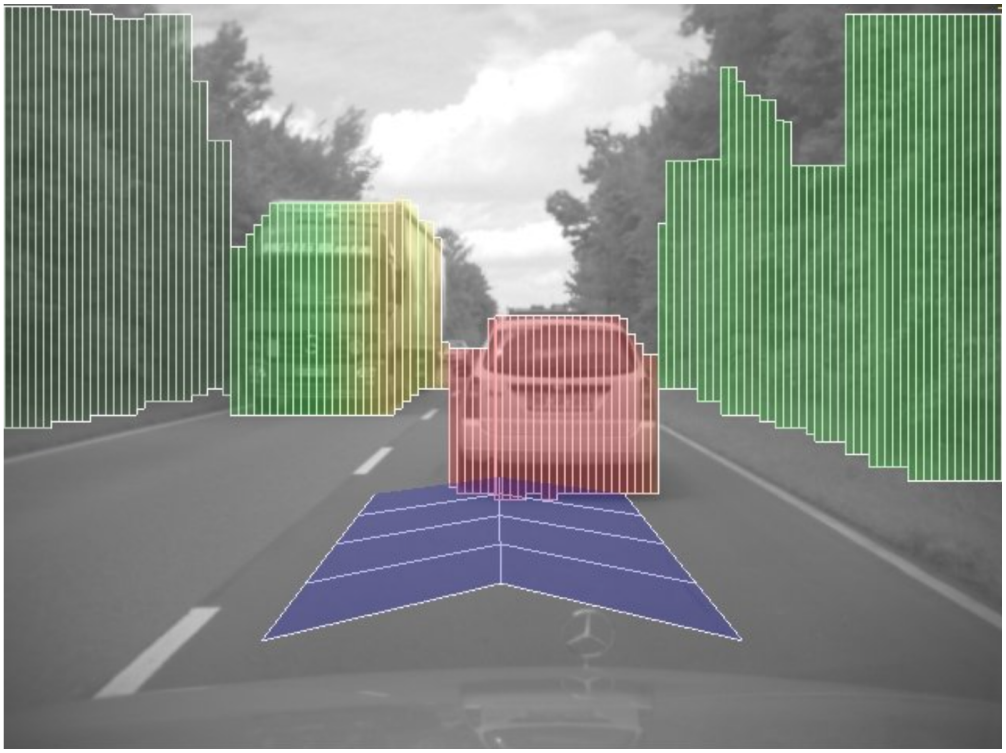


Figure 7.1.: Stixel representation created from raw camera image [Badino, Franke, and Pfeiffer 2009]. The vertical rectangles inform the ego vehicle about the free space in front of the ego vehicle.

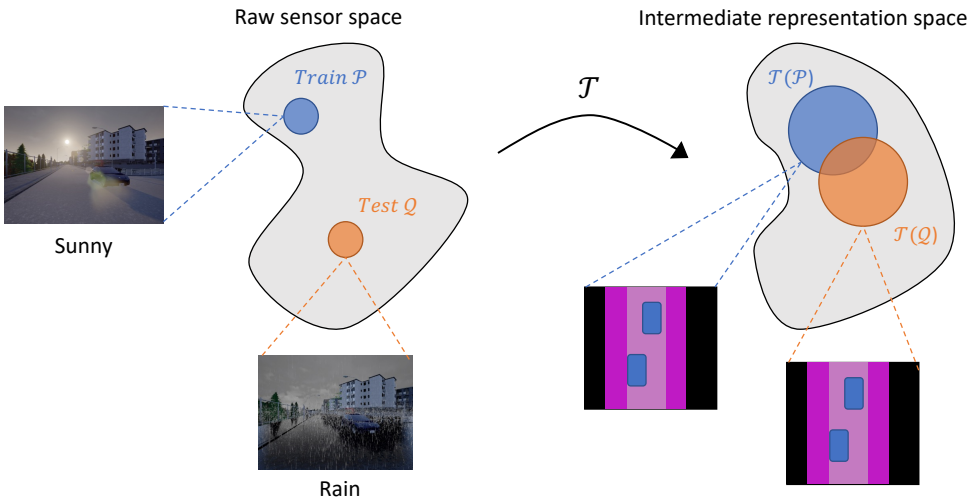


Figure 7.2.: An illustration of how an intermediate representation improves generalization. In this example, RGB images are transformed to a semantic BEV image which makes the intermediate representation generalize well. However, the weather information is useful and need to be encoded in another way.

7.1. The Circogram environment representation

An autonomous vehicle can be thought of as a system that operates in a plane limited by 3D obstacles compared to drones operating in the 3D-space. This plane simplifies the real world where there exist non-touching overlaps like bridges and tunnels. The space that the vehicle operates in is called a *collision corridor*, which is the 3D space limited on the bottom by the road plane, and on the top by a virtual plane parallel to the road plane in the height of the ego vehicle. Figure 7.3 illustrates four examples of this space. Any obstacle in the space is represented by a red vertical bar that goes over the complete height of the corridor. It does not matter whether it is only a point object somewhere in that vertical bar or an object with a certain vertical extension along this bar, such as a pole. The remaining of this section is based on my preproject report [Aaslund 2021].

One intermediate environment representation that is based on this simplification of the environment is the Circogram by Klose and Mester 2019. The Circogram is created by sending out N rays from the center of the vehicle. For each of these rays, there are two points of interest. The endpoint of a Circogram ray, which is given by the presence of an object at any height between the road surface and the car height. The endpoint can be considered as a virtual Stixel, which is described by Badino, Franke, and Pfeiffer 2009. Badino, Franke, and Pfeiffer 2009 represent the area where the vehicle can drive as free space and obstacles as sticks with different colors. The second point of interest is the point that represents the outer hull of the vehicle. This information is crucial if one wishes to drive in a narrow alley without scratching the vehicle surface or bumping into something. Figure 7.4 illustrates the Circogram representation.

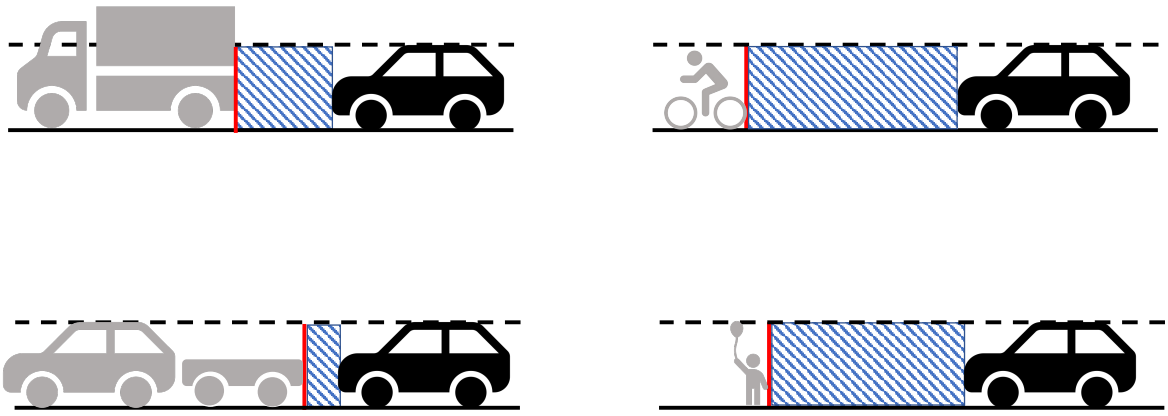


Figure 7.3.: Collision corridors created from different objects. The red vertical bars illustrates the objects that the black ego vehicle can collide with. The blue striped box show the free space in front of the ego vehicle.

Figure 7.4.: Creation of a Circogram from a semantic BEV image. The image to the left illustrates a semantic BEV and red simulated rays. The image to the right shows the ego vehicle and the hitpoints illustrated with red dots.

This representation of the environment is a sparse and compact abstraction of the environment, but it is a sufficient representation of the driving situation. This means that the process of obtaining the best driving action at any time will require less computing power and hence also be faster. One limitation of this approach is that it might be difficult to drive if the roads do not contain curbs or highways with several lanes. Also, a different system is needed to handle the traffic rules.

It is important to use the distance from the surface of the vehicle to the objects around the vehicle,

instead of the distance from the center of the vehicle. The ego vehicle is defined to be centered in origo and pointing in $\frac{\pi}{2}$. Then, the surface coordinates in the xy-plane are calculated by,

$$\begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{cases} \begin{bmatrix} \text{sgn}(\cos(\alpha)) \cdot w/2 \\ x \cdot \tan(\alpha) \end{bmatrix} & \text{if } w \cdot |\sin(\alpha)| < \ell \cdot |\cos(\alpha)| \\ \begin{bmatrix} \text{sgn}(\sin(\alpha)) \cdot \ell/2 \\ y/\tan(\alpha) \end{bmatrix} & \text{else} \end{cases} \tag{7.1}$$

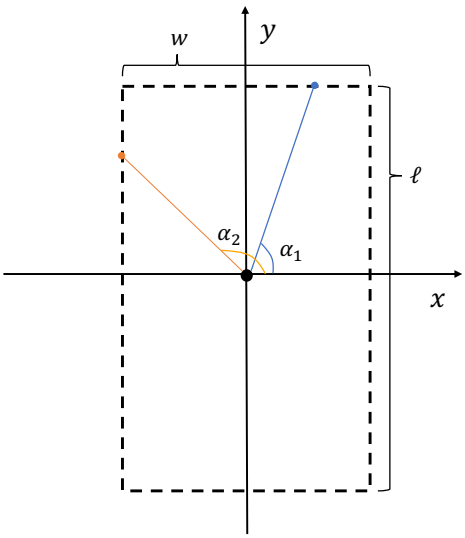


Figure 7.5.: Illustration of how the boarder points of the ego vehicle are calculated. The blue point in the end of the blue line is calculated with the first part of the picewise function in Equation (7.1), and the orange point in the end of the orange line is calculated with the second part of the equation.

where w is the width of the vehicle, ℓ is the length of the vehicle, and α is the angle between the direction of the vehicle and the LiDAR ray that hit the object. This is further illustrated in Figure 7.5. These calculations make the assumption that the vehicle is a rectangle. The distance of the Circogram ray is calculated by,

$$d_i = \sqrt{(x_i - x_s)^2 + (y_i - y_s)^2} \tag{7.2}$$

where i is one of the Circogram rays, and d_i is the distance of the circogram ray.

This approach could be extended by adding more available information to the Circogram rays. The extension of the static Circogram representation will be called the dynamic Circogram. The static Circogram is extended by adding a semantic class and the relative velocity with respect to the ego

vehicle. By combining Circograms close in time, it is possible to obtain an more expressive representation. From two Circograms it is possible to obtain the relative velocity vector of the surrounding objects. In this thesis it is assumed that perfect sensors exists, and the true instantaneous velocity of the other moving objects are used. The dynamic Circogram is illustrated in Figure 7.6.

Figure 7.6.: Creation of a dynamic Circogram from a semantic BEV image and velocity information of other moving objects. The image to the left illustrates a semantic BEV and red simulated rays. The image to the right illustrates a dynamic Circogram. The arrows illustrates the relative velocity, and the points that do not have an arrow have the same velocity as the ego vehicle.

After this transformation the Circogram representation consists of points $p \in \mathrm{R}^4$ on the form $(d_i, v_{x,i}, v_{y,i}, c_i)$. The same information can also be described as (d_i, s_i, ξ_i, c_i), where s_i is the speed, and ξ_i is the angle between the velocity vector $\mathbf{v_i}$ and the direction of the ego vehicle which is defined as $\frac{\pi}{2}$. This whole intermediate representation will be in the space $\mathrm{R}^{4\times N}$, where N is the number of Circogram rays.

The dynamic Circogram can be used to understand how the surrounding objects are moving relative to the ego vehicle. If the relative velocity vector is close to $\mathbf{0}$, it means that the object move with the same speed and in the same direction as the ego vehicle. A high relative speed means that the distance between the object and the ego vehicle is changing quickly. The distance is growing or decressing depending on the orientation of the relative velocity vector. This means that the ego vehicle needs to pay more attention to the objects that has a velocity direction towards the ego vehicle, and it does not need to worry about the objects that have a relative velocity pointing away. The dynamic Circogram makes it easy to understand which objects that can become conflicting, and hence put the attention at these points. One issue with the static Circogram is that it is not possible to differ between a vehicle in fort of the ego vehicle and a wall. The dynamic Circogram can easily differ between these two scenarios since they have different relative speed compared to the ego vehicle.

7.1.1. Creation of the dynamic Circogram

There are different ways to produce the Circogram. It is possible to create the Circogram from a semantic BEV or from a semantic LiDAR. These two approaches have different drawbacks. A semantic BEV is created in CARLA by attaching a semantic camera above the ego vehicle, as explained in section 6.6. This setup creates a problem if there exist objects that cover the road. These objects can be low hanging trees or lamp posts. These objects can be assigned to an own class

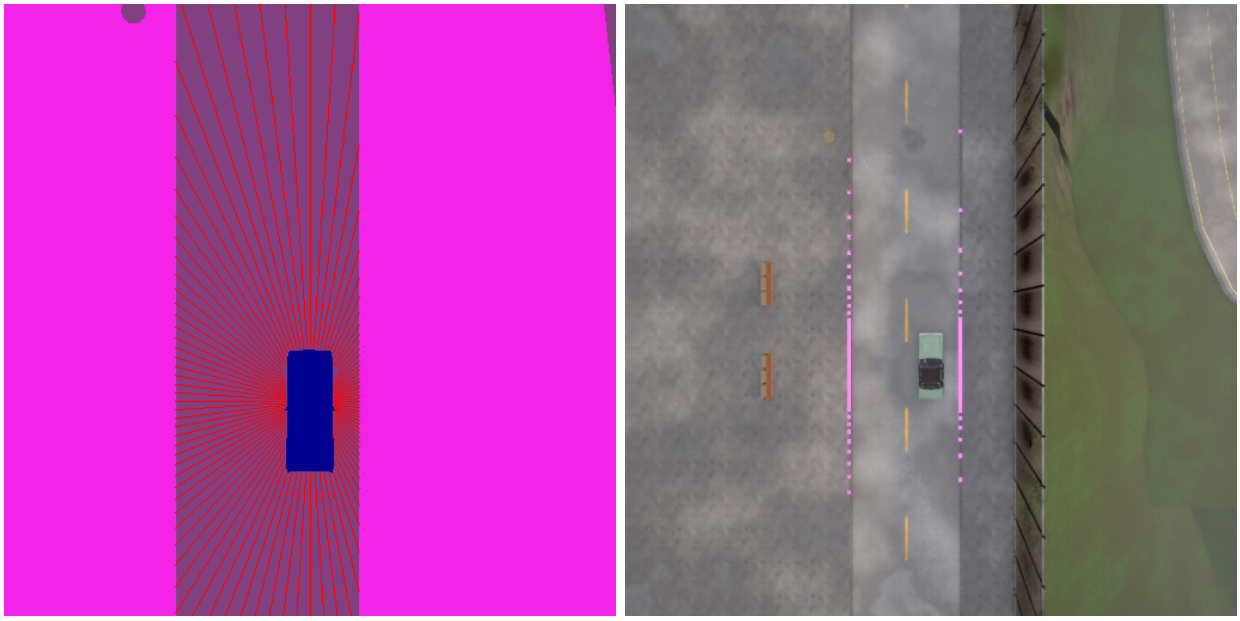
Figure 7.7.: Creation of a Circogram ray from several LiDAR rays. The red lines to the left illustrates laser rays, and the striped blue area close to the car with the LiDAR illustrates a deadspot that the LiDAR cannot detect. The upper red line in the image to the left hits the other car. The image to the right shows the resulting Circogram ray from above the cars.

called "do not care" and handled accordingly or removed from the simulation. The second approach is to use LiDAR data. The LiDAR sensor sends out rays in N directions above the ego vehicle. The closest LiDAR ray which hits something in the relevant height above the ground corridor causes an endpoint of the corresponding Circogram ray. The Circogram ray is obtained from this LiDAR ray by backprojecting it onto the 2D plane. A limitation of this approach is the deadspots that are created close to the vehicle, which are areas the LiDAR rays do not detect. Figure 7.7 illustrates the process of generating a Circogram from a simulated LiDAR. The figure to the left shows a LiDAR that hits a different vehicle, and the figure to the right shows the resulting Circogram ray. One deadspot is illustrated with the blue striped area close to the car with the LiDAR in Figure 7.7.

In this thesis, the recolored semantic BEV described in section 6.6 is used to create the Circogram. This sensor returns BEV images $\mathcal{F} \in [0, 255]^{rc \times 3}$, with r and c being respectively the number of rows and columns of the image. An affine transformation that transforms coordinates in the ego vehicle coordinate frame to pixel coordinates is used to find the pixel of the center of the vehicle. This affine transformation is found by the least squares method and is further described in the Appendix B. The center of the vehicle in the ego coordinate frame is transformed into a pixel o_p. A virtual circle in the pixel coordinate frame with o_p as the center and a sufficient radius is selected (that makes the whole image fit inside the circle). This circle is used to distribute the Circogram rays. Each Circogram ray is sent out of o_p with a certain angle θ_i. The angles θ_i are discretized in the interval $[0, 2\pi)$ with step $\Delta\theta = \frac{2\pi}{n}$.

The length of the Circogram rays is found by traversing each line of pixels from o_p to each of the equally distributed points on the circle. The line of pixels between o_p and one point of the circle is found by the Bresenhams line algorithm. These lines are traversed one at a time, and the pixel where the Circogram ray starts is the first pixel that corresponds to a drivable color. The pixel where the Circogram ray ends is the first pixel after the start pixel, which does not correspond to a drivable class. This is denoted as a *hit point*. This class will be the class that corresponds to this Circogram ray. The length of the Circogram ray is then calculated by transforming the pixel coordinates to the ego vehicles coordinate frame, and calculating the euclidean distance between the points. The Circogram at this point is a vector $v \in \mathrm{R}^{n \times 2}$). This matrix contains the distance of each Circogram ray l_i (i.e. the distance from the car to a hit point) having a certain angle θ_i, and classes of the endpoints. The rays that hit the edge of the image is assigned an own class called *free direction*, and the length is set to 30 m.

The SafeRide system assumes perfect sensors to avoid data preprocessing of subsequent Circograms to obtain the relative velocity. The endpoint of the Circogram ray is the coordinates of the location where the ray hits another object. It is possible to retrieve additional information about this point



(a) Illustration of how the static part of the Circogram is created in CARLA. The red lines are the simulated Circogram rays. They are only drawn to better illustrate how the static part of the Circogram is created.

(b) The resulting hitpoints after sending out the simulated Circogram rays are marked with pink boxes. The simulated rays that did not hit anything before the image edge are not marked. This image is taken further away than the image used for the Circogram to better show the hitpoints.

Figure 7.8.: Illustration of how the static part of the Circogram is created in CARLA.

in CARLA. The additional information is retrieved by looping through all the actors (objects that interfere with the simulation in CARLA) and checking if the Circogram ray has hit it. We retrieve which actor which is located at this location. Once this is done, we extract the velocity and angular velocity this object had during the last tick. The algorithm below further explains this process.

Algorithm 1 Static to dynamic Circogram

```
procedure GenerateDynamicCircogram(staticCircogram, actorList)
    for point in staticCircogram do
        for actor in actorList do
            if point in actor.get_boundingbox() then
                point[velocity] ← actor.get_velocity()
                point[angular_velocity] ← actor.get_angular_velocity()
                break
            end if
        end for
    end for
end procedure
```

The relative velocity of the endpoint of the Circogram ray with respect to the start of the Circogram ray can be calculated from the instantaneous motion vector for both points, the vector $\mathbf{l_i}$, and the momentary rotation rate $\boldsymbol{\omega_i}$ and $\boldsymbol{\omega_s}$. The hit point rotates with orbital angular velocity $\boldsymbol{\omega_i}$ about its center of rotation in a coordinate frame F_i which itself rotates with a spin angular velocity $\boldsymbol{\omega_s}$ with respect to an external frame F_s, we can define $\boldsymbol{\omega} = \boldsymbol{\omega_i} + \boldsymbol{\omega_s}$ to be the composite orbital angular velocity vector of the point about its center of rotation with respect to F_s. The $\boldsymbol{\omega_i}$ and $\mathbf{v_i}$ will be

$\mathbf{0}$ when the Circogram ray hits a static object. CARLA uses the left hand rule to determine the direction of the angular velocity. The velocity of the hit point of the Circogram ray i is defined as $\mathbf{v_i}$, and the instantaneous motion vector of the ego vehicle is defined as v_s. The relationship between the velocities when the reference frame is rotating is given by,

$$\mathbf{v_i} = \mathbf{v_s} + \boldsymbol{\omega} \times \mathbf{l_i} + \mathbf{v_{i/s}} \tag{7.3}$$

Then the relative velocity vector $\mathbf{v_{i/s}}$ is calculated with,

$$\mathbf{v_{i/s}} = \mathbf{v_i} - \mathbf{v_s} - \boldsymbol{\omega} \times \mathbf{l_i} \tag{7.4}$$

7.2. The potential field environment representation

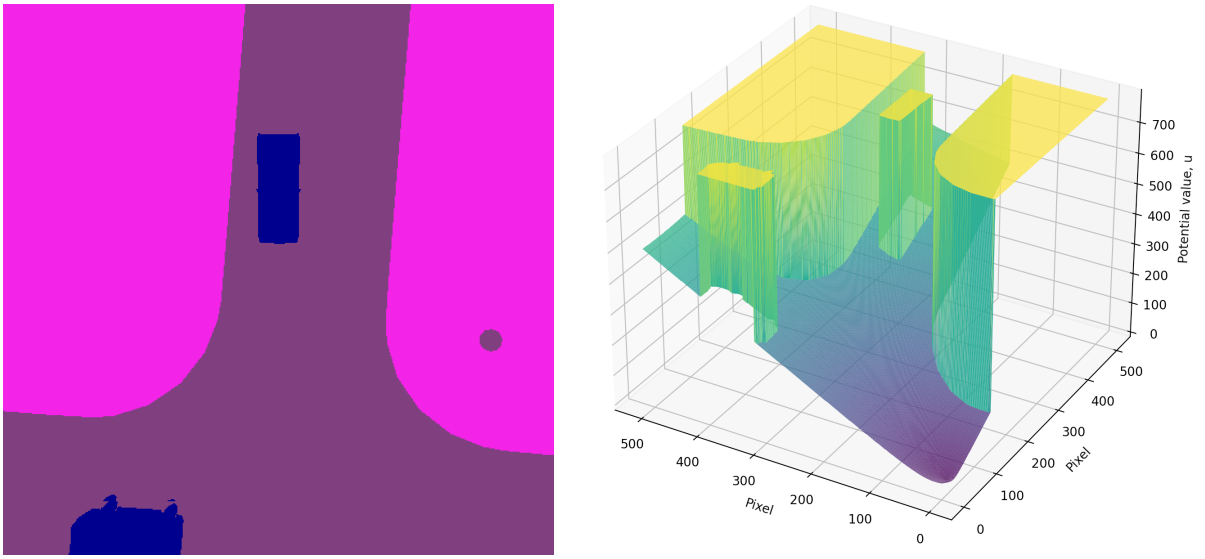
The dynamic Circogram environment representation includes the necessary information for driving safely, but it does not contain any information about where the autonomous vehicle should drive. The dynamic Circogram is created from the recolored BEV sensor explained in section 6.6. A *temporary goal pose* is defined to be a location in the recolored BEV that the autonomous vehicle want to arrive. The temporary goal pose works as an attractor on the ego vehicle.

One common way to encode the attraction to a position in the two-dimensional plane is to use a potential field approach. This approach is based on a potential function which can be interpreted as energy, and the gradient of the potential function as a force. The potential function will guide the vehicle as if it is a particle in a gradient field. Usually obstacles will have the opposite effect of the temporary goal pose, obstacles will act as a repulsive force. The value of the potential function will be a surface if it is plotted in three dimensions. One widely used algorithm to approach the temporary goal pose is to use gradient decent. This potential field approach is a rather commonly used approach in robot motion planning [Koren and Borenstein 1991]. In SafeRide , the vehicle is considered as a dynamical system which is a subject to non-holonomic constraints. This means that the vehicle cannot change the motion direction and speed instantaneously. This makes the planning harder, and simple approaches like gradient decent cannot be used. Although the potential field cannot directly be used to find a plan, it still contains valuable information that will be used in section 9.5.

7.2.1. Creation of potential field from semantic BEV in CARLA

The potential field is created from the recolored BEV sensor, which is also used to create the dynamic Circogram. Each pixel in the recolored BEV sensor contains a categorical value, which will be transformed to a real positive value. The high-level plan provided by CARLA (explained in subsection 9.2.1) is used to find the temporary goal pose. The location of the temporary goal pose is defined to be the last point in the high-level plan that is inside the recolored BEV.

The creation of the potential field starts by extracting the coordinates of the temporary goal pose. This coordinate is the starting point of a recursive approach to calculate the distances to the other pixels in the free space. The distance between two neighbor pixels is set to one. The neighbors to the neighbors are traversed recursively in a breadth first manner, until all the pixels in the free space have a potential value. The remaining pixels are assign a high value The result is a positive valued potential field in image form which contain information about how the ego vehicle can reach the temporary goal pose. Figure 7.9 shows a recolored image that is rotated 180 degrees, and the corresponding potential field when the temporary goal pose is down to the right in the recolored



(a) One example output from the custom recolored BEV sensor. The image is rotated 180 degrees to easily see the connection to the potential field.

(b) An output from the custom recolored BEV sensor and the corresponding potential field. The temporary goal pose is down to the right, which corresponds to the ego vehicle taking a left turn in the intersection.

Figure 7.9.: A recolored BEV from the custom sensor together with the corresponding potential field.

BEV. Figure 7.9a include a manhole cover on the sidewalk which CARLA label as a road. Figure 7.9b shows that the manhole cover gets the same potential value although it is labeled as drivable in the recolored BEV, and hence only the pixels in the free space are traversed.

Repulsive forces are not included since the autonomous vehicle should be able to drive close to static objects. However, repulsive forces could be added in the velocity direction of other moving objects. This is not done in this system since that information already is encoded in the dynamic Circogram, and therefore redundant.

8. Physical Vehicle Motion Models

Estimating the motion of a vehicle is a crucial requirement for intelligent vehicles. The benefit of having a motion model is that it is possible to predict how the vehicle will move based on the throttle and steering input. It is however difficult to create a model that precisely can describe how a vehicle will move in the real world. Therefore, it is often necessary to make assumptions to come up with simplified models that are approximately correct.

There exists many different motion models for vehicles, and they differ in the number of assumptions that they make. The physics based motion models can be divided into kinematic motion models and dynamical motion models. Kinematic models describe a vehicle's motion based on the mathematical relationship between the parameters of the movement, without considering the forces that affect the motion. Schubert, Richter, and Wanielik 2008 test and analyze different motion models, and argue for which settings the different models are appropriate. In kinematic models, all the forces are neglected, and geometry and trigonometric identities are used to derive the models. The dynamical models extend the kinematic models by also taking forces into consideration. Vehicles are governed by complex physics, and therefore dynamic models can get extremely large and involve many internal parameters of the vehicle. These motion models exists as differential equations, and corresponding time-discrete models.

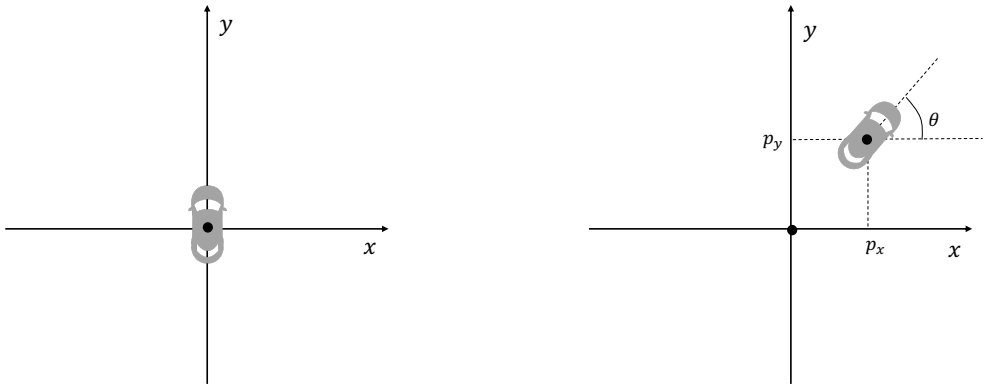
Two different coordinate frames are used to explain the physical motion model. Figure 8.1 shows the vehicle coordinate frame and the world coordinate frame. The vehicle coordinate frame is defined to have origo at the current center of mass of the vehicle, and the y-axis pointing in the same direction as the longitudinal axis of the vehicle. In the world coordinate frame, the position of the vehicle is described by a point (p_x, p_y), and an orientation angle θ. The important symbols that are used in this chapter is listed and explained in Table 8.1.

In this project we split up the complete vehicle model into a steering model and a drive train model. The drive train model describe the resulting velocity as being effected by the throttle (or brake) pedal, whereas the steering model describe the trajectory of the car, dependent on steering input while moving forward. The overview of the complete model is illustrated in Figure 8.2. The vehicle model leads to the same trajectory in space, no matter what the velocity is, if the steering angle is coupled to the traveled distance on that trajectory, or if this course is traveled fast or slowly. The different parts of the physical vehicle motion model will be described in this chapter.

8.1. Constant turn rate and velocity model

The simplest class of kinematic motion models describes the motion based on the velocity v and the orientations angle of the vehicle θ as inputs. The orientation angle θ is the instantaneous orientation of the vehicle, defined as the angle between the x-axis and the longitudinal axis of the vehicle.

One of these models is called *constant turn rate and velocity model (CTRV)*, and it is illustrated in Figure 8.3. This model uses the current position of the center of mass to the vehicle D with coordinates (p_x, p_y), the current velocity v, the orientation angle θ, and the rate of change of orientation $\dot{\theta}$ (also called turn rate) to compute the next vehicle state vector $\mathbf{x}$. Due to this setup,



(a) Vehicle frame. (b) World frame.

Figure 8.1.: The different coordinate frames that are used to create the physical motion models, and the relevant symbols.

the vehicle runs on a circular course with a rotation center C. This model makes the assumption that the turn rate is constant $\ddot{\theta} = 0$, and that the velocity is constant, $\dot{v} = 0$. This means that the output will be a new position (p_x, p_y) and a new orientation angle θ. The state vector $\mathbf{x}$ for this model is defined as,

$$\mathbf{x} = \begin{bmatrix} p_x \\ p_y \\ v \\ \theta \\ \dot{\theta} \end{bmatrix}, \tag{8.1}$$

and hence the change rate of the state vector is written as,

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{v} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ 0 \\ \dot{\theta} \\ 0 \end{bmatrix} \tag{8.2}$$

The input of the orientation angle θ is assumed to be a linear function of time between two time instants t_k and t_{k+1}, and the difference between them is Δt. From this it follows that $\theta_{k+1} =$

Symbol	Definition
p_x	x-coordinate of the center of mass of the vehicle in the world frame.
p_y	y-coordinate of the center of mass of the vehicle in the world frame.
θ	The orientation in the world frame defined as the angle between the x-axis and the longitudinal axis of the vehicle.
$\dot{\theta}$	The change in orientation of the vehicle in the world frame.
ϕ	Angle between the direction of the tire and the longitudinal axis of the vehicle.
β	The slip angle, defined as the angle between the velocity of the center of mass and longitudinal axis of the vehicle.
z	Steering input.
r	Throttle input.
b	Brake input.
C_r	The point of the rotation center in the world frame.
A	The point of the front tire in the world frame.
B	The point of the rear tire in the world frame.

Table 8.1.: Important symbols in the physical motion model. The symbols are defined throughout this chapter and reused in the remaining chapters.

$\int_{t_k}^{t_{k+1}} \theta_k + \dot{\theta}_k(t - t_k)dt$. The time-discrete model then becomes,

$$\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{x}_k + \int_{t_k}^{t_{k+1}} \begin{bmatrix} v_k \cos(\theta_k + \dot{\theta}_k(t-t_k)) \\ v_k \sin(\theta_k + \dot{\theta}_k(t-t_k)) \\ 0 \\ \dot{\theta}_k \\ 0 \end{bmatrix} dt = \mathbf{x}_k + \begin{bmatrix} \int_{t_k}^{t_{k+1}} v_k \cos(\theta_k + \dot{\theta}_k(t-t_k))dt \\ \int_{t_k}^{t_{k+1}} v_k \sin(\theta_k + \dot{\theta}_k(t-t_k))dt \\ 0 \\ \int_{t_k}^{t_{k+1}} \dot{\theta}_k dt \\ 0 \end{bmatrix} \\
&= \mathbf{x}_k + \begin{bmatrix} v_k \left[\frac{1}{\dot{\theta}_k} \sin(\theta_k + \dot{\theta}_k(t-t_k)) \right]_{t_k}^{t_{k+1}} \\ v_k \left[-\frac{1}{\dot{\theta}_k} \cos(\theta_k + \dot{\theta}_k(t-t_k)) \right]_{t_k}^{t_{k+1}} \\ 0 \\ \Delta t \cdot \dot{\theta}_k \\ 0 \end{bmatrix} = \mathbf{x}_k + \begin{bmatrix} \frac{v_k}{\dot{\theta}_k} \Big(\sin(\theta_k + \dot{\theta}_k \Delta t) - \sin(\theta_k) \Big) \\ \frac{-v_k}{\dot{\theta}_k} \Big(\cos(\theta_k + \dot{\theta}_k \Delta t) - \cos(\theta_k) \Big) \\ 0 \\ \Delta t \cdot \dot{\theta}_k \\ 0 \end{bmatrix}
\end{aligned} \tag{8.3}$$

CTRV assumes that there is no functional relation between the velocity v and the turn rate $\dot{\theta}$. As a consequence, disturbed turn rate measurements can change the orientation angle of the vehicle even if it is not moving [Schubert, Richter, and Wanielik 2008]. To overcome this issue, one can model the functional relation between the velocity and the turn rate by the steering angle.

8.2. Constant steering angle and velocity

One model that uses the steering angle ϕ is called *constant steering angle and velocity model (CSAV)*, also called the bicycle model in the literature [Rajamani 2012]. This model simplifies a 4 wheeled

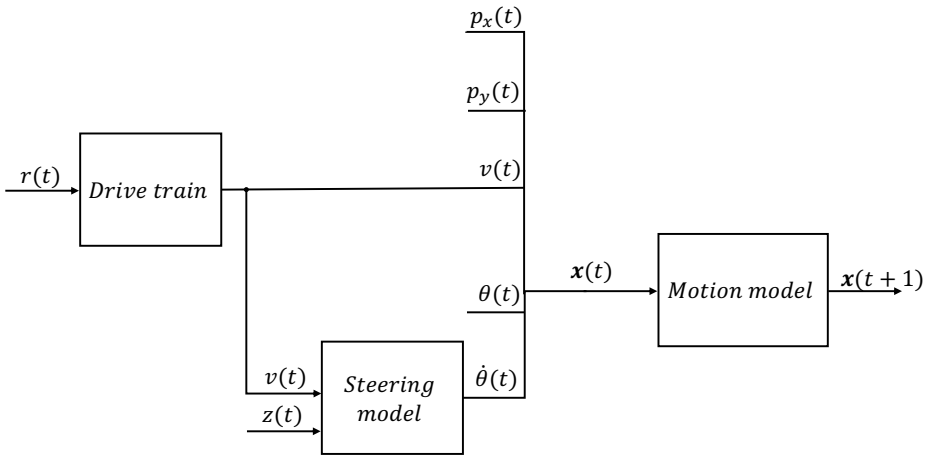


Figure 8.2.: Overview of the complete motion model. The complete motion model takes the current position (p_x, p_y), the orientation θ, the throttle input r, and the steering input z, and produces a new dynamic vehicle state.

vehicle by merging the front tires together and the rear tires together, hence it is sometimes called the bicycle model. The mechanical construction of the vehicle is simplified to only contain two tires that are mounted in a distance L. An illustration of this model and the variables used to define the model is provided in Figure 8.4. It is necessary to define several angles to describe the motion of this model. The front tire can be steered and takes an steering angle ϕ relative to the longitudinal axis of the vehicle. The orientation angle θ is the instantaneous orientation of the vehicle, defined as the angle between the x-axis and the longitudinal axis of the vehicle. The center of mass D of the vehicle has the coordinates (p_x, p_y) and is located on the longitudinal axis and has the distance ℓ_f from the front tire and the distance ℓ_r from the rear tire, thus $L = \ell_f + \ell_r$. D serves as the origin (center) of the vehicle coordinate system, and is the point that we aim to describe how will move. Therefore, both translation information as well as rotation information is given with respect to the center of gravity. Since the front tire is allowed to change direction, the velocity vectors at each tire may have different directions. This means that the velocity vectors at point A and point B are in the same direction as the front tire and the rear tire respectively. From this it follows that the velocity vector of the center of mass D to the vehicle may be different from the direction of the velocity vector of the front tire. As Figure 8.4 shows, both road contact points of the tires as well as the center of gravity of the vehicle move on concentric circles around the center of rotation C_r with different radii. A closer look reveals furthermore, that the tangential angles of the motion in these 3 points are different. Therefore, another angle, the slip angle β is introduced. Given these 3 points, the relation between the slip angle β (which is the one which finally characterizes the motion of the vehicle) and the steering angle ϕ at the front tire is found by geometry and Figure 8.4.

By applying the sine rule to triangle ACD and BCD, and assuming that the steering angle at the

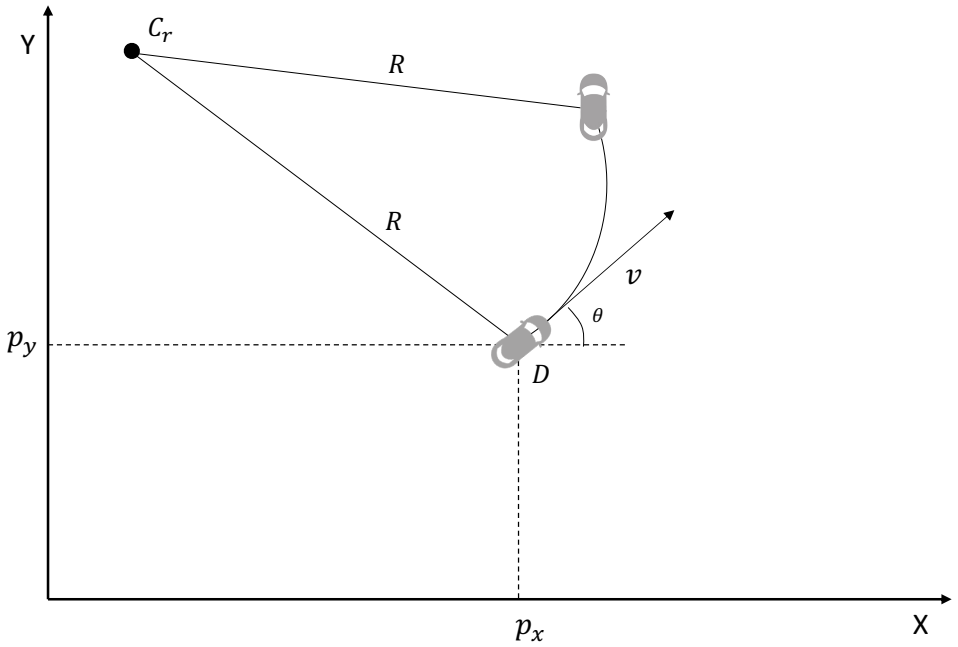


Figure 8.3.: Illustration of the symbols that are used in the constant turn rate and velocity model.

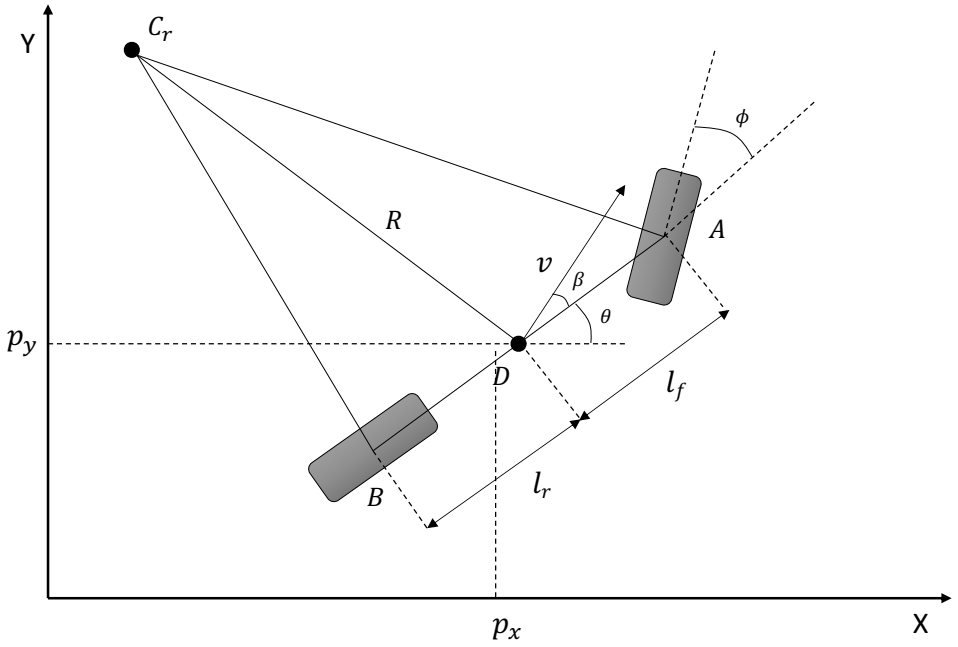


Figure 8.4.: Illustration of the of a simplified vehicle model, where the front tires and the rear tires are respectively merged from two to four tires. The variables used in the derivation of the equations in this section are defined here.

rear tire is 0,

$$\frac{\sin(\phi - \beta(\phi))}{\ell_f} = \frac{\sin\left(\frac{\pi}{2} - \phi\right)}{R} \tag{8.4}$$

$$\frac{\sin(\beta(\phi))}{\ell_r} = \frac{\sin\left(\frac{\pi}{2}\right)}{R} = \frac{1}{R} \tag{8.5}$$

By applying $\sin(\alpha_1 - \alpha_2) = \sin(\alpha_1)\cos(\alpha_2) - \sin(\alpha_2)\cos(\alpha_1)$ we get,

$$\frac{\sin(\phi)\cos(\beta(\phi)) - \sin(\beta(\phi))\cos(\phi)}{\ell_f} = \frac{\cos(\phi)}{R} \tag{8.6}$$

Now we multiply both sides of Equation (8.6) with $\frac{\ell_f}{\cos(\phi)}$ and get,

$$\tan(\phi)\cos(\beta(\phi)) - \sin(\beta(\phi)) = \frac{\ell_f}{R} \tag{8.7}$$

The slip angle $\beta(\phi)$ is obtained by multiplying Equation (8.7) with ℓ_r and subtracting Equation (8.5) multiplied with ℓ_f,

$$\beta(\phi) = \arctan\left(\frac{l_r \tan(\phi)}{L}\right) \tag{8.8}$$

Figure 8.5 illustrates how realistic values of ϕ is transformed into β-values.

The CSAV model uses the steering angle ϕ instead of the turn rate $\dot{\theta}$ in the state variable,

$$\mathbf{x} = \begin{bmatrix} p_x \\ p_y \\ v \\ \theta \\ \phi \end{bmatrix}, \tag{8.9}$$

The radius R of the vehicle path is constant due to the constant steering angle ϕ between two timesteps, then the turn rate is equal to the angular velocity of the vehicle. This results in $\dot{\theta} = \frac{v}{R}$. The course angle of the vehicle is given by $\theta + \beta(\phi)$. Instead of the steering angle ϕ being a direct input to the differential equation system, it is first converted into the slip angle β. By decomposing the velocity v, we obtain the change rate of the state space formulation,

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{v} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v \cdot \cos(\beta(\phi) + \theta) \\ v \cdot \sin(\beta(\phi) + \theta) \\ 0 \\ \frac{v\cos(\beta(\phi))}{L}\tan(\phi) \\ 0 \end{bmatrix}. \tag{8.10}$$

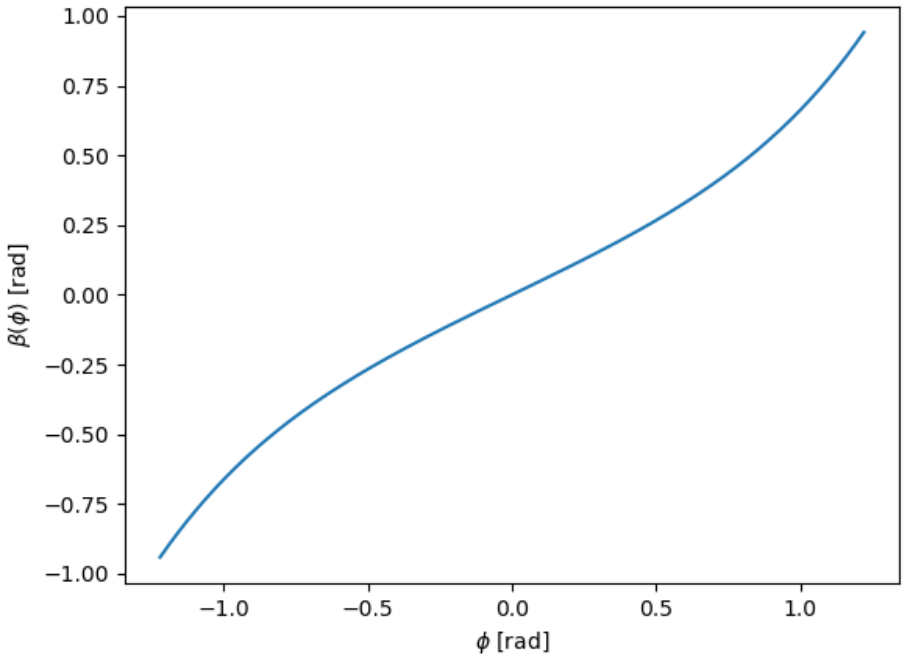


Figure 8.5.: Graph of the relation between the steering input and the slip angle. The slip angle $\beta(\phi)$ for steering inputs $\phi \in [-1.22, 1.22]$ rad.

A derivation of these equations is given by Kong et al. 2015. Recall that both the velocity v and the steering angle ϕ inputs are assumed to be constant. These are the only input variables since the turn rate $\dot{\theta}$ only depends on the steering angle ϕ. Equation (8.10) is integrated over Δt with these

assumptions on the inputs, then the time-discrete model becomes,

$$
\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{x}_k + \int_{t_k}^{t_{k+1}} \begin{bmatrix} v \cdot \cos(\beta(\phi) + \theta) \\ v \cdot \sin(\beta(\phi) + \theta) \\ 0 \\ \dfrac{v \cos(\beta(\phi))}{L} \tan(\phi) \\ 0 \end{bmatrix} = \mathbf{x}_k + \begin{bmatrix} \int_{t_k}^{t_{k+1}} v_k \cos(\beta(\phi)_k + \theta_k + \dot{\theta}_k(t - t_k)) \\ \int_{t_k}^{t_{k+1}} v_k \sin(\beta(\phi)_k + \theta_k + \dot{\theta}_k(t - t_k)) \\ 0 \\ \int_{t_k}^{t_{k+1}} \dfrac{v_k \cos(\beta(\phi)_k)}{L} \tan(\phi_k) \\ 0 \end{bmatrix} \\
&= \mathbf{x}_k + \begin{bmatrix} v_k \left[\dfrac{1}{\dot{\theta}_k} \sin(\beta(\phi_k) + \theta_k + \dot{\theta}_k(t - t_k)) \right]_{t_k}^{t_{k+1}} \\ v_k \left[-\dfrac{1}{\dot{\theta}_k} \cos(\beta(\phi_k) + \theta_k + \dot{\theta}_k(t - t_k)) \right]_{t_k}^{t_{k+1}} \\ 0 \\ \Delta t \dfrac{v_k \cos(\beta(\phi_k))}{L} \tan(\phi_k) \\ 0 \end{bmatrix} \\
&= \mathbf{x}_k + \begin{bmatrix} \dfrac{v_k}{\dot{\theta}_k} \Big(\sin(\beta(\phi_k) + \theta_k + \dot{\theta}_k \Delta t) - \sin(\beta(\phi_k) + \theta_k) \Big) \\ \dfrac{-v_k}{\dot{\theta}_k} \Big(\cos(\beta(\phi_k) + \theta_k + \dot{\theta}_k \Delta t) + \cos(\beta(\phi_k) + \theta_k) \Big) \\ 0 \\ \Delta t \cdot \dfrac{v_k \cos(\beta(\phi_k))}{L} \tan(\phi_k) \\ 0 \end{bmatrix}
\end{aligned}
\tag{8.11}
$$

8.3. Dynamic bicycle model

This section is largely based on my preproject report [Aaslund 2021]. The previous kinematic models assumes that there are no slip between the tire and the road. This is an assumption that only holds for low speeds, and with high friction between the tire and the road. This assumption will be broken in this research, since slippery roads will lead to lower frictions between the tires and the road. Therefore, it is important to have a motion model that allows this to happen. The dynamic bicycle model is an extension of the bicycle model, where one allows slip between the tires and the road. This means that the velocity direction and the tire direction do not need to be the same.

Although this model relaxes one assumption, it still is based on several assumptions. The model assumes that the vehicle's motion is restricted to the xy-plane. The vehicle is restricted to be a rigid body. Only lateral tire forces, generated by a linear tire model. This assumption is okay as long as we are in the area where the tire model is linear. One also assumes small steering angle, which is valid in high speed but might be broken when driving in city centres and parking. The final assumption is that it assumes constant longitudinal velocity.

All derivations below are based on Pepy, Lambert, and Mounier 2006. To derive a state space representation of the dynamics, we need to describe the lateral dynamics, the orientation dynamics, and the tire force. Figure 8.6 define the symbols that are used for the derivations. To derive the

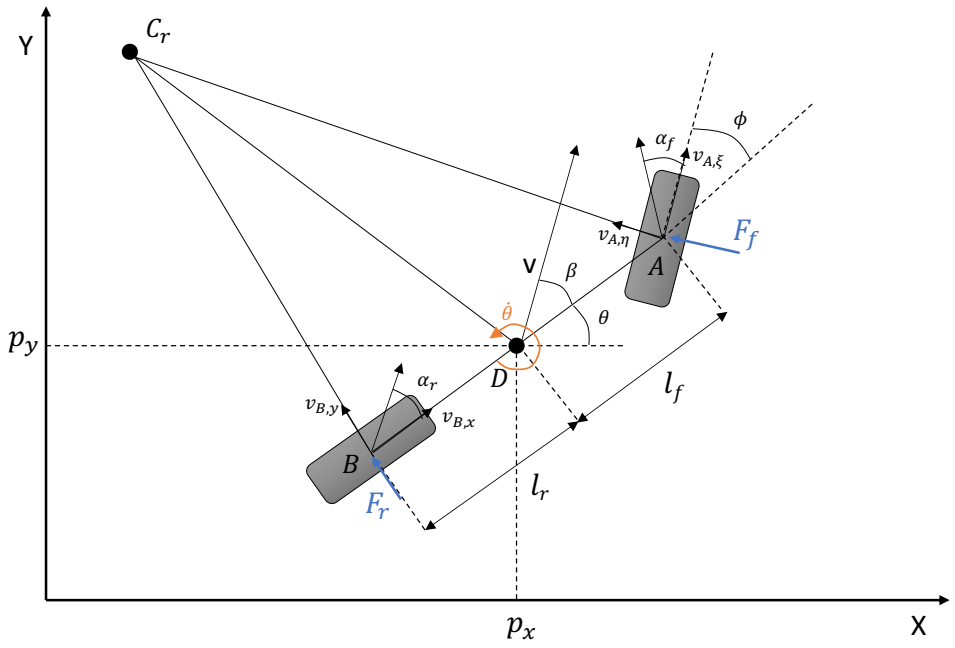


Figure 8.6.: Illustration of the dynamic bicycle model, and definition of variables used in the derivation of the equations.

lateral dynamics, we need to start by Newtons second law,

$$\begin{aligned} ma_y &= \sum_i F_{y,i} = F_r + F_f \cos(\phi) \approx F_r + F_f \\ a_y &= \dot{v}_y + \dot{\theta} v_x \\ \Rightarrow m(\dot{v}_y + \dot{\theta} v_x) &= F_r + F_f \end{aligned} \tag{8.12}$$

where F_r and F_f are respectively the rear tire force and the front tire force that we assume follow a linear model. This means that the forces can be written as $F_r = c_r \alpha_r$ and $F_f = c_f \alpha_f$. c_r and c_f are the cornering stiffness coefficients for front and rear tires. Since we are only considering motions in the xy-plane and a rigid body, we only need to consider one moment of interia, $I_z = \int_B (x^2 + y^2) dm$. To derive the orientation dynamics, we need to use the angular momentum principle,

$$\begin{aligned} I_z \ddot{\theta} &= \sum_i M_i = -\ell_r F_f + \ell_f F_f \cos(\theta) \\ \Rightarrow I_z \ddot{\theta} &= -\ell_r F_f + \ell_f F_f \end{aligned} \tag{8.13}$$

The only unknown variables in Equation (8.12) and Equation (8.13) are the tire forces. Since we assume the tire forces are linear we get,

$$\begin{aligned} F_r &= -c_r \alpha_r \approx -c_r \tan(\alpha_r) = -c_r \frac{v_{B,y}}{v_{B,x}} \\ F_f &= -c_f \alpha_f \approx -c_f \tan(\alpha_f) = -c_f \frac{v_{A,\eta}}{v_{A,\xi}} \end{aligned} \tag{8.14}$$

where $v_{B,x} = v_x$, $v_{B,y} = v_y - \dot{\theta}\ell_r$, $v_{A,x} = v_x$, $v_{A,y} = v_y + \dot{\theta}\ell_f$. In the front coordinate system we get $v_{A,\xi} = v_{A,x}\cos(\theta) + v_{A,y}\sin(\theta)$ and $v_{A,\eta} = -v_{A,x}\sin(\theta) + v_{A,y}\cos(\theta)$. By plugging in these into Equation (8.14) we obtain,

$$\begin{aligned} F_r &= -c_r \frac{v_{B,y}}{v_{B,x}} = -c_r \frac{v_y - \dot{\theta}\ell_r}{v_x} \\ F_f &= -c_f \frac{v_{A,\eta}}{v_{A,\xi}} = -c_f \frac{-v_x\theta + (v_y + \dot{\theta}\ell_f)}{v_x + (v_y + \dot{\theta}\ell_f)\theta} \approx c_f\theta - c_f \frac{(v_y + \dot{\theta}\ell_f)}{v_x} \end{aligned} \tag{8.15}$$

In the last approximation, we have used that $v_x >> (v_y + \dot{\theta}\ell_f)\theta$. By inserting the tire forces into Equation (8.12) and Equation (8.13) we get the state space formulation,

$$\begin{bmatrix} \dot{v}_y \\ \dot{\phi} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} -\frac{c_r + c_f}{m v_x} & 0 & \frac{c_r\ell_r - c_f\ell_f}{m v_x} - v_x \\ 0 & 0 & 1 \\ \frac{c_r\ell_r - c_f\ell_f}{I_z v_x} & 0 & \frac{c_r\ell_r^2 - c_f\ell_f^2}{I_z v_x} \end{bmatrix} \begin{bmatrix} v_y \\ \phi \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} \frac{c_f}{m} \\ 0 \\ \frac{c_f}{I_z}\ell_f \end{bmatrix} \phi \tag{8.16}$$

The state space representation can be integrated and used to extend the kinematic models described in section 8.2 to simulate the vehicle dynamics.

8.4. System identification for the vehicle motion models

The driver can only influence the behavior of the vehicle by pressing the gas pedal and turning the wheel. However, the motion models in section 8.1, section 8.2, and section 8.3 use the speed and the change in orientation as inputs instead of the actuator inputs. Hence it is necessary to model the relationship between the actuator inputs, and the inputs to the vehicle models. This is done by simple physical modelling together with system identification in the SafeRide system. subsection 8.4.1 describes the longitudinal drive train, and subsection 8.4.2 describes the relationship between the steering input and the inverse radius of the arc the vehicle is currently driving.

There exists two different ways to perform system identification; active and passive system identification. The input signal is selected or designed in active system identification, and the resulting output from the system is observed. The relationship between the input and the output signal can then be described with a transfer function. The passive system identification is when we observe a system in operation, record the input signal as well as the output signal, and infer the relationship between the two of them from a long sequence of input signals. The passive system identification also leads to a transfer function in the same way as the active system identification. The passive system identification is not done on the basis of a self designed or a self selected signal but just on the basis of what the system does in operation. One disadvantage with the active system identification is that it requires appropriate road network, which might be hard to create. For example it is necessary to have a big parking lot to perform the system identification of the steering model. If the system is a simple proportional system, or just a simple non-linear input - output relation, it might be simpler to estimate it using the passive approach. The longitudinal drive train in subsection 8.4.1 is based on active system identification, and the steering input to inverse radius model uses passive system identification.

8.4.1. Longitudinal drive train model

The throttle is the actuator which influences the velocity, and hence the SafeRide system need to be able to related the current speed together with a throttle input to a new velocity. This observation creates the need for a model that transforms the throttle to velocity. The motion models use the previous velocity to predict the next state, and therefore it is more appropriate to have model that transforms throttle to velocity instead of a model that transforms throttle to acceleration. This model is called a longitudinal drive train model, and it will be derived by first modelling the relation between the velocity and throttle with physics, and then system identification will be performed to estimate the constants in the equation. There also exists several complete physical models that relates the throttle to velocity or acceleration. These physical models require several constants from the motor dynamic. Such constants are hard to obtain in CARLA, and thus simple physical modelling together with system identification is used to create the model.

An analysis of the forces that works on a vehicle can be used to easier perform the system identification of the longitudinal drive train. Let us assume that the thrust, the air resistance and the friction resistance are the only forces that works on the vehicle. The throttle is, in first order approximation, proportional to the motor moment, and the motor moment on the drive axle is converted by the tire into a linear thrust. The thrust and the counterforces, exerted by the tire friction on the road and the air resistance are in balance when the acceleration is zeros. Let F be the thrust, $r \in (0, 1)$ be the throttle value, then $F(r) = \alpha \cdot r(t)$ with $F_{max} = \alpha$. The air resistance and the road friction are approximated to be proportional to the speed $v(t)$. Let furthermore M be the mass of the vehicle. Then by Newton´s second law,

$$\sum_i F_i = M\dot{v}(t) \tag{8.17}$$

and thus,

$$\dot{v}(t) = \frac{(\alpha \cdot r(t) - \beta(\phi) \cdot v(t))}{M} \tag{8.18}$$

From this it follows that the throttle model can be approximated to follow a first order differential equation on the form,

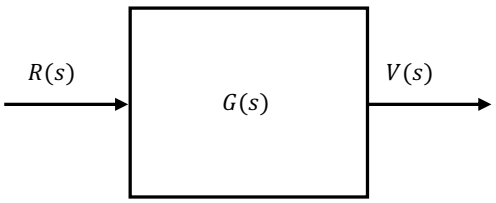
$$r(t) = A\dot{v}(t) + Bv(t) \tag{8.19}$$

This assumption is valid since a step function input does not make the output to overshoot, and will result in one zero in the transfer function. Taking the Laplace transformation of the differential equation we obtain,

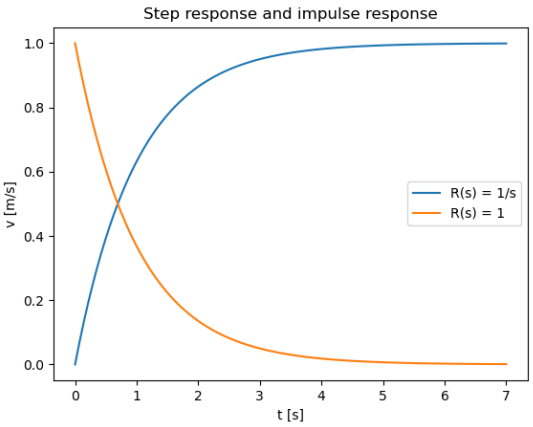
$$\begin{aligned} R(s) &= A(sV(s) - v(0)) + BV(s) \\ &= V(s)(As + B) - Av(0) \end{aligned} \tag{8.20}$$

Te transfer function can then be calculated if we assume that $v(0) = 0$.

$$G(s) = \frac{V(s)}{R(s)} = \frac{1}{As + B} = \frac{K_L}{\tau_L s + 1} \tag{8.21}$$



(a) Block diagram of the drive train plant.



(b) The system response to a impulse and step signal at $t = 0$, and $K_L = 1$ and $\tau_L = 1$

Figure 8.7.: Block diagram and response of the drive train system.

where,

$$K_L = \frac{1}{B} \text{ and } \tau_L = \frac{A}{B}.$$

A block diagram of the longitudinal system model is shown in Figure 8.7a. This means that the output of the system will be the inverse Laplace transform of $V(s) = R(s)G(s)$. $R(s) = 1$ is selected to calculate the impulse response of the system, and $R(s) = \frac{1}{s}$ is selected to calculate the step response of the system. The two responses in the time domain are respectively,

$$v(t) = \mathcal{L}^{-1}\left(1 \cdot \frac{K_L}{\tau_L s + 1}\right) = \mathcal{L}^{-1}\left(\frac{K_L}{\tau_L} \cdot \frac{1}{s + \frac{1}{\tau_L}}\right) = \frac{K_L}{\tau_L} \cdot e^{\left(-\frac{1}{\tau_L}\right)t}, \tag{8.22}$$

$$v(t) = \mathcal{L}^{-1}\left(\frac{1}{s}\frac{K_L}{\tau_L s + 1}\right) = \int_0^t \frac{K_L}{\tau_L} \cdot e^{(-t/\tau_L)} dv = K_L\left(1 - e^{(-t/\tau_L)}\right) \tag{8.23}$$

These two responses of the system are plotted in Figure 8.7b.

The time constant of the plant and the proportionality factor have been approximated by evaluating the response of the system to different step values. The proportionality factor corresponds to the gain of the system, and is calculated from steady state signals r_{ss} and v_{ss},

$$K_L = \frac{\Delta v}{\Delta r} = \frac{v_{ss} - v_0}{r_{ss} - r_0} \tag{8.24}$$

The time constant regulates how quick the response of the system is. The smaller the constant, the faster the system approaches the steady state. The value of this constant corresponds to the time at which the output signal reaches the 63.2% of the steady state value from the moment that the

step input changes. It is calculated by setting $t = \tau_L$ for a step input with value r,

$$v(\tau_L) = r \cdot K_L \left(1 - e^{\left(-\frac{1}{\tau_L}\right)\tau_L}\right) = r \cdot K_L \left(1 - e^{-1}\right) = 0.632 \cdot r \cdot K_L \tag{8.25}$$

This means that τ_L is the time where the velocity v is $0.632 \cdot u \cdot K_L$.

Several experiments have been performed to estimate the proportionality factor K_L and the time constant τ_L. The experiments are conducted by placing the vehicle on a long straight road. The vehicle is a car with automatic gearing[1], and it is initially standing still. The throttle r is pressed to some test value r_i. The car accelerates until it reaches some stable end velocity v_{ss} after some time. The location, the velocity, and the acceleration are saved every $\frac{1}{30}s$.

Step signals from 0.1 to 1.0 have been used to see how the plant react. The open loop response for all the step functions are shown in Figure 8.8. The plots show that the system looks like a first order system, but the time constants and the proportionality factors seems to change with the throttle r.

[1]The gear shift time in CARLA is set to 0 to simplify the dynamics.

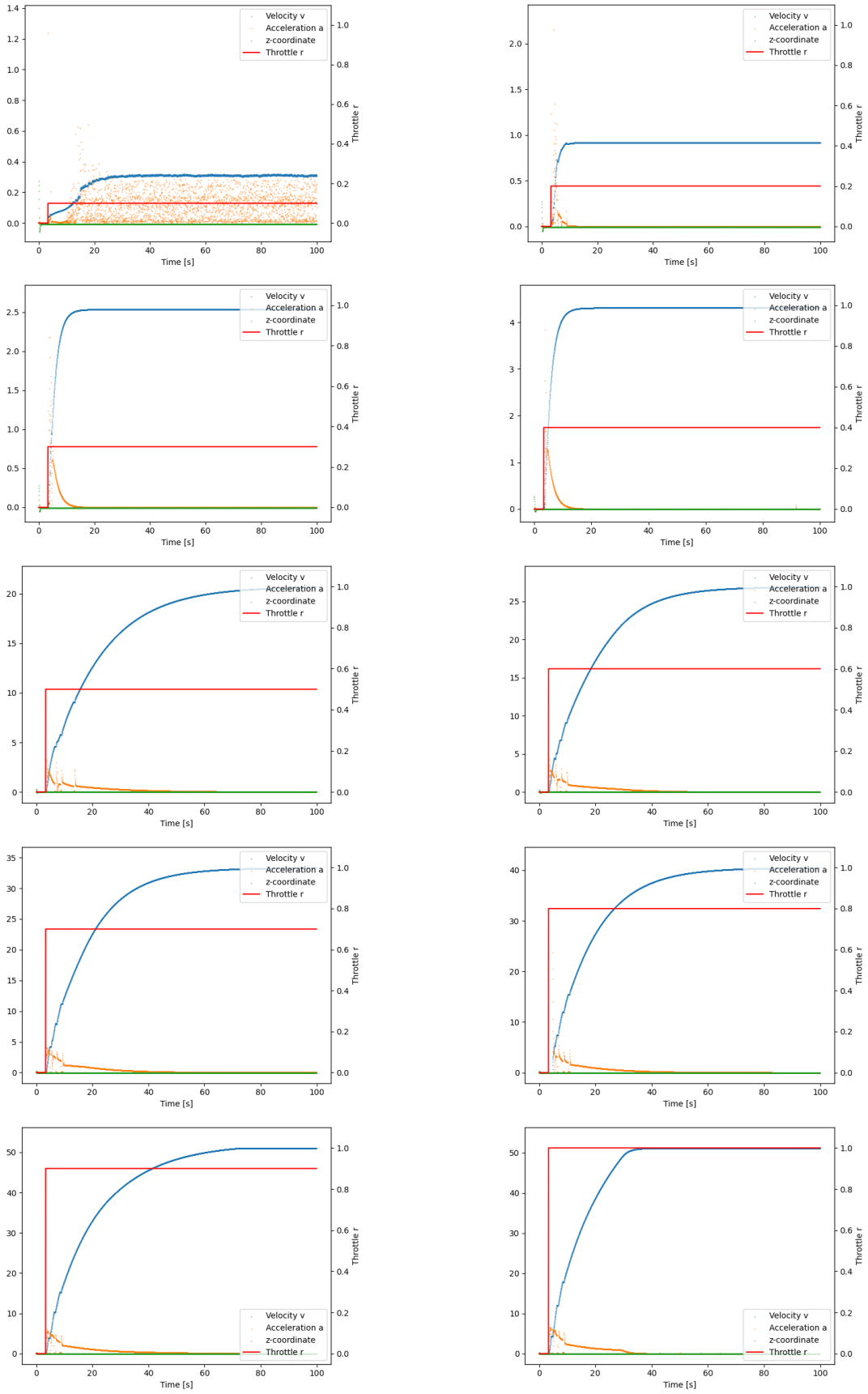


Figure 8.8.: Analysis of the drive train for different step functions for the throttle input. The blue line is the velocity, the orange dots are the acceleration, the red line is the step input, and the green line is the z.coordinate.

System identification			
Step input	v_{ss} [m/s]	K_L	τ_L [s]
0.1	0.3213	3.2130	11.5333
0.2	0.9172	4.5860	1.7333
0.3	2.5230	8.4100	2.5000
0.4	4.3075	10.7689	2.5000
0.5	20.7613	41.5226	17.8000
0.6	26.9109	44.8515	16.2333
0.7	33.3450	47.6357	14.8667
0.8	40.4055	50.5069	14.8000
0.9	51.0440	56.7156	16.0000
1.0	51.1523	51.1523	12.3667

Table 8.2.: The proportionality factor K_L and the time constants τ_L in the drive train for different step inputs of the throttle.

The velocity v looks noisy when the throttle r is 0.1. K_L and τ_L are calculated for each step signal with Equation (8.24) and Equation (8.25), and listed in Table 8.2.

A similar test is performed for the braking system. The vehicle is accelerated to an initial speed, and different step braking inputs b are tested. The initial speed is selected to be 5.56 m/s (20 km/h), 13.89 m/s (50 km/h), and 27.78 m/s (100 km/h). Step signals from 0.1 to 1.0 have been used to see how the plant react. Figure 8.9 shows the result for an initial velocity of 13.89 m/s and a brake input of 0.1 and 1.0. All the experiments showed that the velocity decreases approximately linearly, where the slope depends on the braking input b. This means that the deacceleration from the brake is constant for each braking input b. The deaccelerations from all the experiments are summarized in Table 8.3.

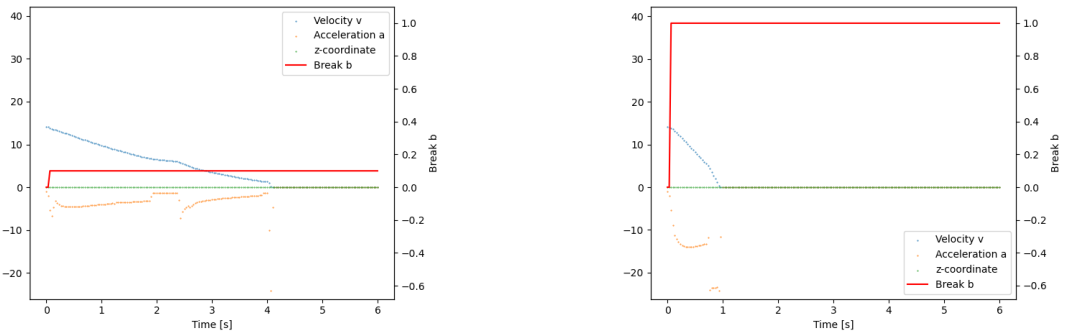


Figure 8.9.: System identification for brake input 0.1 and 1.0 with initial velocity of 13.89 m/s. The blue line is the velocity, the orange dots are the acceleration, the red line is the step input, and the green line is the z.coordinate.

Once again we assume an operating velocity of 13.89 m/s (50 km/h), and perform a linear regression of the deacceleration from 13.89 m/s (50 km/h). Figure 8.10 show the regression of the deacceleration against the brake input for an initial velocity of 13.89 m/s (50 km/h).

The coefficients from the regression are listed in Table 8.3. It is assumed that the braking input b_k is constant function of time between two time instants t_k and t_{k+1}. The following equation is used

Braking deacceleration			
Step input	5.56 [km/h]	13.89 [km/h]	27.78 [km/h]
0.1	6.69 [m/s^2]	3.09 [m/s^2]	3.03 [m/s^2]
0.2	6.94 [m/s^2]	4.00 [m/s^2]	4.04 [m/s^2]
0.3	6.94 [m/s^2]	4.79 [m/s^2]	5.12 [m/s^2]
0.4	6.94 [m/s^2]	6.61 [m/s^2]	5.41 [m/s^2]
0.5	6.94 [m/s^2]	8.17 [m/s^2]	7.72 [m/s^2]
0.6	6.94 [m/s^2]	8.32 [m/s^2]	8.60 [m/s^2]
0.7	7.21 [m/s^2]	8.85 [m/s^2]	9.48 [m/s^2]
0.8	7.21 [m/s^2]	9.45 [m/s^2]	10.29 [m/s^2]
0.9	7.21 [m/s^2]	9.71 [m/s^2]	10.98 [m/s^2]
1.0	7.21 [m/s^2]	10.14 [m/s^2]	11.72 [m/s^2]

Table 8.3.: Braking deacceleration for different initial velocities, and different braking inputs.

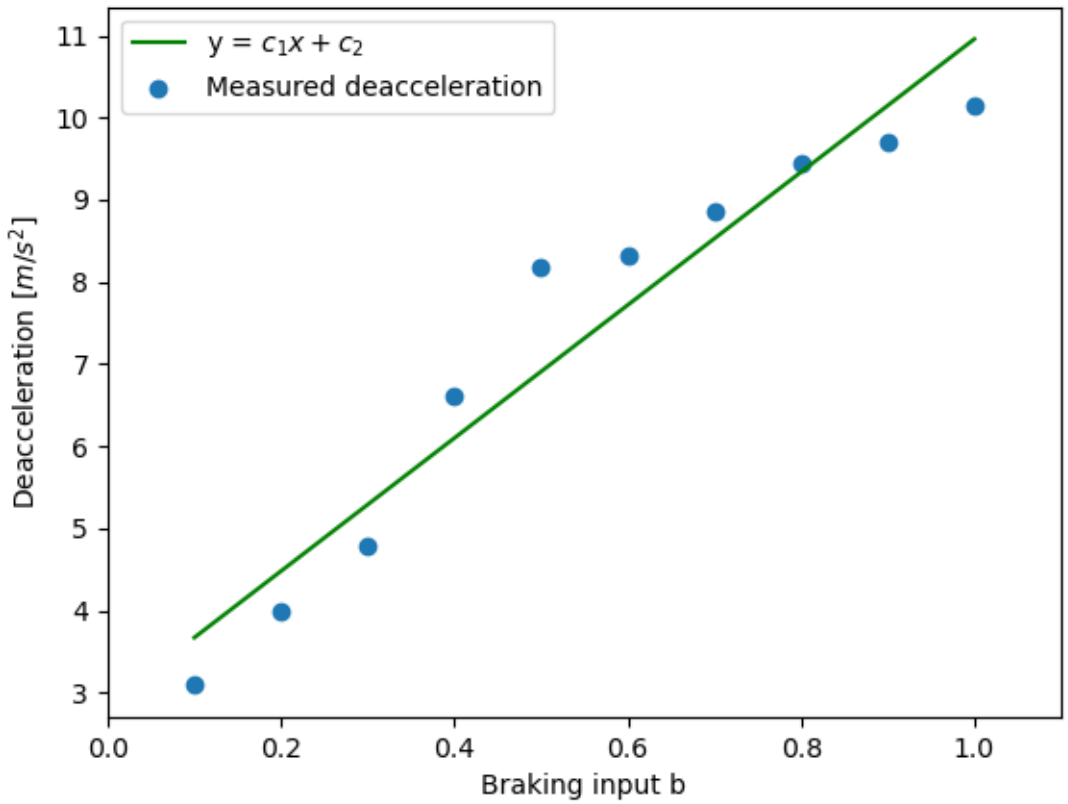


Figure 8.10.: Regression of the braking input against the deacceleration measures from an initial velocity of 13.89 m/s (50 km/h). The blue dots are the measured deacceleration for the corresponding braking input, and the green line is the regression line.

to perform a prediction of the velocity when a braking input is applied

$$v_{k+1} = v_k - (8.1 \cdot b_k + 2.86) \cdot \Delta t. \tag{8.26}$$

Type of data	$\mathbf{c_1}$	$\mathbf{c_2}$	MSE
Initial velocity of 13.89 [m/s]	8.10 [m/s^2]	2.86 [m/s^2]	0.40 [m/s^2]

Table 8.4.: Results from the regression of the braking input against the deacceleration.

This complete the longitudinal drive train to consist of two equations which transforms the throttle or brake together with the current velocity to the next velocity.

8.4.2. Steering to inverse radius model

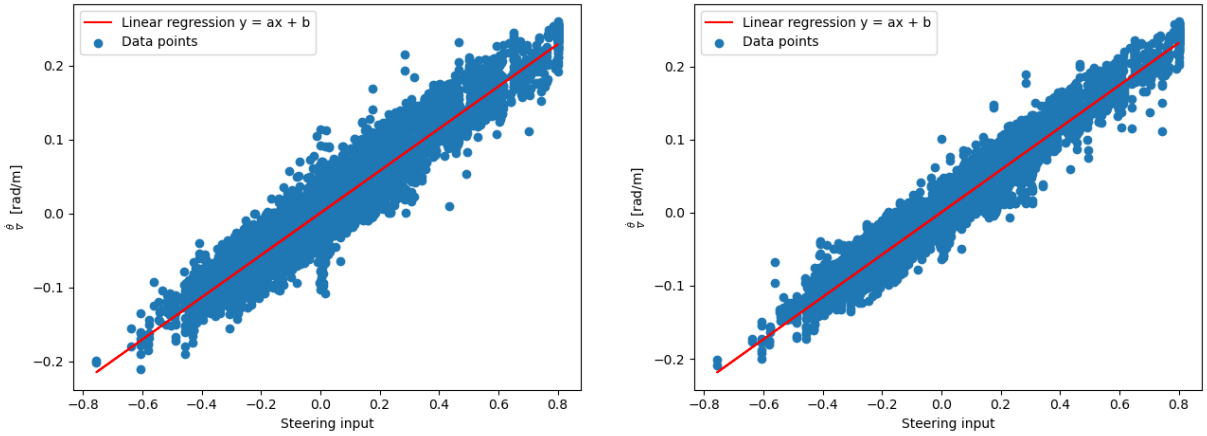
For the steering to inverse radius model, it is appropriate to use a passive system identification. A passive system identification is selected due to the lack of large open area where different steering inputs could be investigated. It is required to model the nonlinerar relation between the steering input ϕ and the radius of the circle that results from the steering input. The radius R is related to the turn rate $\dot{\theta}$ and the velocity v by,

$$\dot{\theta} = \frac{v}{R}. \tag{8.27}$$

The turn rate $\dot{\theta}$ as a function of the steering angle will also depend on the vehicle speed v. Each vehicle in CARLA comes with a maximum steering angle which correspond to an steering input z of 1 or -1. Hence it is necessary to record the orientation angle θ, the steering input z, and the vehicle velocity v. These values are recorded while an autopilot drives randomly around in a town. The town should have a wide range of different curves, such that many different steering maneuvers are used. Town03 in CARLA is selected for the passive system identification of the steering based on this requirement. The resulting dataset will contain points (θ_i, z_i, v_i), where $i = 1, \ldots, N$. The time between each recorded point is $\Delta t = \frac{1}{20}$ since the simulation is run synchronously. We assume that the turn rate $\dot{\theta}$ is linear between two timesteps, hence it can be estimated from two subsequent orientation angles,

$$\dot{\theta}_i = \frac{\theta_{i+1} - \theta_i}{\Delta t} \tag{8.28}$$

The preprocessed dataset contain $(\dot{\theta}_i, z_i, v_i)$, where $i = 1, \ldots, N-1$. The system identification problem of the steering is now reduced to a regression problem. The goal is to find a function f that maps steering inputs z to $\dot{\theta}$ that can be used in the vehicle motion model. Since the turn rate $\dot{\theta}$ depend on both the steering input z and the velocity v, we can divide the turn rate $\dot{\theta}$ by the velocity v to remove this dependency. By doing this, we are actually finding a function that maps the steering input z to the inverse of the radius R of the circle path. After this, we are left by finding a function f that maps the steering input z to $\frac{\dot{\theta}}{v} = \frac{1}{R}$. Thus, $g(z) = \frac{1}{R}$, and $v \cdot g(z) = v \cdot \frac{1}{R} = \dot{\theta}$.



(a) Regression using the ground truth turn rate $\dot{\theta}$

(b) Regression using the calculated turn rate from subsequent orientation measures θ

Figure 8.11.: Regression of the measured steering input against the inverse radius of the curve $\dot{\theta}$ divided by the velocity v.

The data in Figure 8.11 looks to have a linear relationship. Therefore, a linear regression is used to approximate the relationship between the steering input z and the inverse radius $\frac{\dot{\theta}}{v}$. The coefficients from the regression are in Table 8.5. It is expected that the function should be odd, since $z = 0$ should result in a line with an infinite radius. Since $a_2 \approx 0$, and the argument above, a_2 is selected to be 0. Therefore,

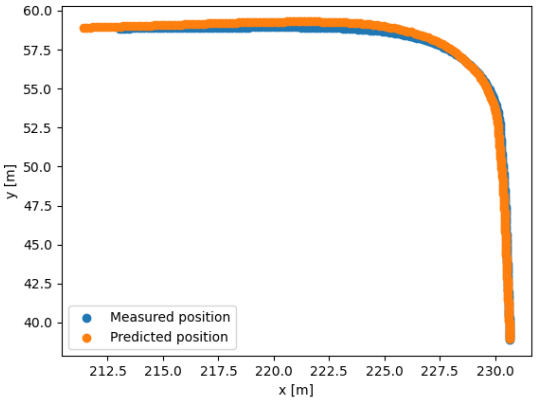
$$\frac{\dot{\theta}}{v} = a_1 \cdot z \tag{8.29}$$

Type of data	$\mathbf{a_1}$	$\mathbf{a_2}$	MSE
Using ground truth turn rate	0.2850 [rad/m]	0.0005 [rad/m]	12.3208e-5 [rad/m]
Using calculated turn rate	0.2897 [rad/m]	0.0005 [rad/m]	7.3438e-5 [rad/m]

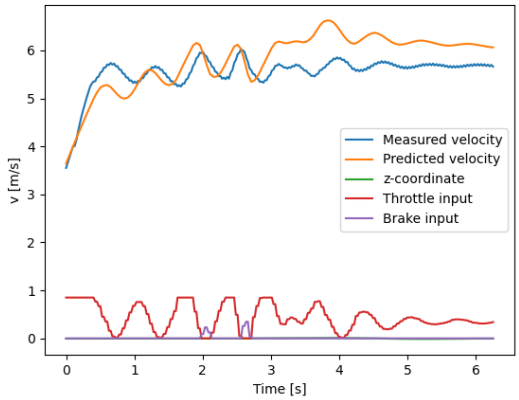
Table 8.5.: Results from the regression of the steering input against the yaw rate $\dot{\theta}$ divided by the velocity v, which is the same as the inverse radius.

8.5. Validation of the physical motion model

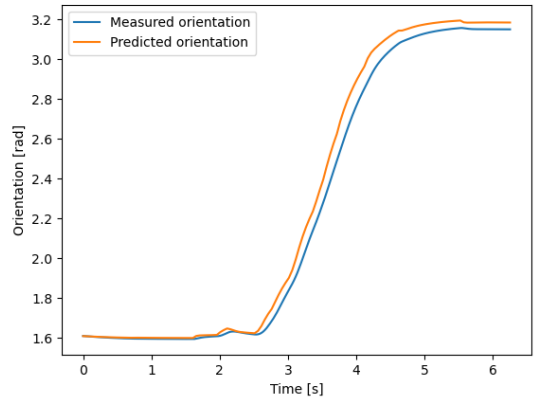
The complete physical motion model is compared against the motion of an autopilot in CARLA when a sequence of steering inputs z, throttle inputs r, and braking inputs b are applied. The complete physical motion model takes the same inputs that are applied to the CARLA vehicle and predicts the next states $\mathbf{x}$. Different driving scenarios are tested (acceleration from 0, stopping, and different turns). The measured states and the predicted states for one of the driving scenarios are compared in Figure 8.12. Figure 8.12c shows orientations above π radians, although the orientation has been defined to be $\theta \in [-\pi, \pi]$. The removal of the discontinuity is explained in Appendix D. This validates the design of the physical motion model, and shows that is approximates the motion of the used vehicle in CARLA.



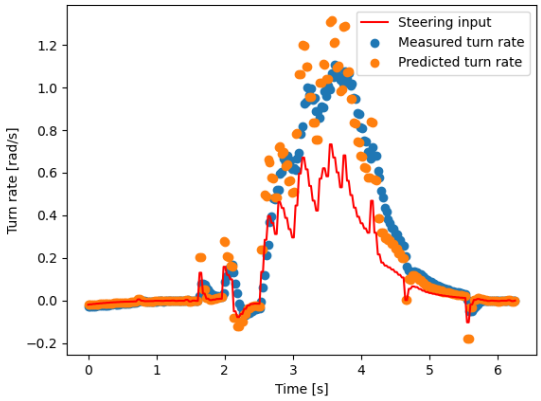
(a) Measured and predicted position.



(b) Measured and predicted velocity. The red line is the throttle input, and the purple line in the brake input.



(c) Measured and predicted orientation together with the input signals that affect the velocity.



(d) Measured and predicted turn rate together with the steering input z. The red line is the steering input.

Figure 8.12.: Comparison of motion model and measured motion of a CARLA vehicle. The orange lines and scatter plots are the predicted data from the motion model while the blue lines and scatter plots are the measured data.

The change in orientation $\dot{\theta}$ is found by multiplying the inverse radius with the current velocity. This value will be one of the inputs in the models described in section 8.1, section 8.2, or section 8.3.

9. The Planning Component

A plan is a loosely defined term, and it is therefore appropriate to start this chapter with a concrete definition of the term. SafeRide uses two different types of plans, short-term plans and high-level plans, and both of these types of plans will be defined. A short-term plan is a hallucinated description of motion over time. This means the position in the world frame (p_x, p_y) of the center of gravity to the ego vehicle, the orientation in the world frame θ, and the velocity v as a function of time. A short-term plan is a sequence of p_x, p_y, θ, and v separated with small time intervals since discrete time is considered. A high-level plan is not as specific as a short-term plan, it includes information about how to reach the final goal location without considering the dynamics of the ego vehicle and surrounding objects. The high-level plan is used to create navigation instructions.

The planning component consists of high-level planning, navigation, control action sequence generation, short-term plan generation, and short-term plan selection. These activities contribute to different levels of the plan, and the result is a short-term plan that the ego vehicle follows for a short period of time before the short-term plan is updated. This chapter contains several defined names, and these names are presented in Appendix A together with a short explanation.

9.1. Expectations of a good autonomous vehicle

A sufficient planning component will lead to an autonomous vehicle that obeys some general expectations. These expectations are listed below.

1. The autonomous vehicle should follow the high-level plan if there do not exist sharp curves or other conflicting objects.
2. The autonomous vehicle should pass conflicting static objects that block the high-level plan and hence deviate from the high-level plan.
3. The autonomous vehicle will deviate from the high-level plan in sharp curves since the high-level plan does not consider the holonomic constraint of the vehicle.
4. The autonomous vehicle should slow down before sharp curves to stay in the correct lane.
5. The autonomous vehicle should stop if the whole driveable area is blocked.

These expectations will be used to validate the planning component and ensure that the autonomous vehicle works well.

9.2. High-level planning between two locations

Drivers use navigation systems to create high-level plans that explain how it is possible to drive from one location to another location. Vehicles are mainly transportation tools used for going from one position to another. For an autonomous vehicle to drive to a goal location, it is necessary to obtain a high-level plan that explains where to go at intersections, which lane to use, and the speed limit. The high-level plan needs to be updated while the vehicle drives along the route. A deviation from

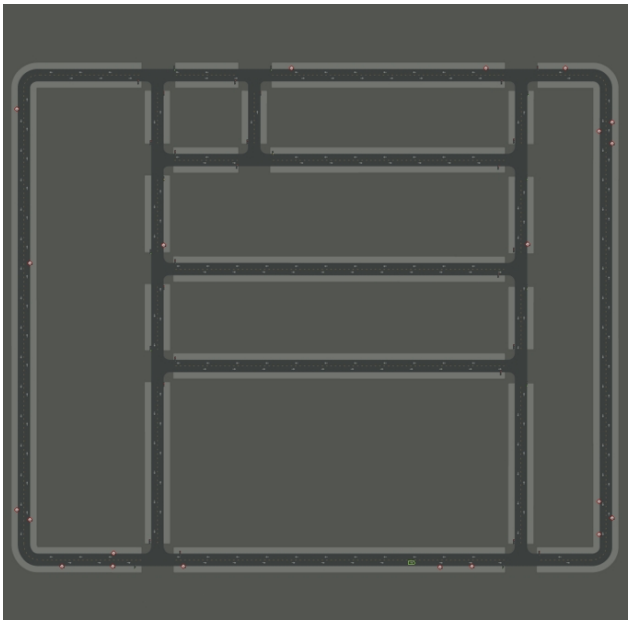


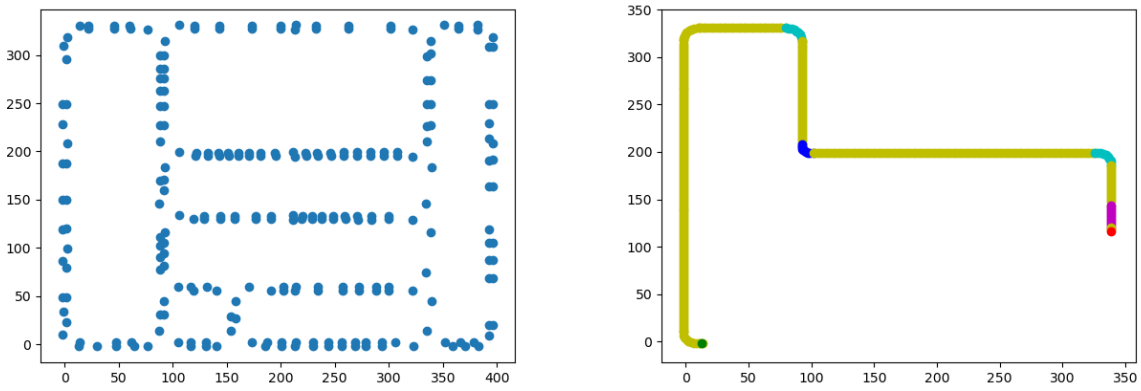
Figure 9.1.: An overview of the road network in Town01 in Carla. The road network consists of T-junctions and sharp curves.

the high-level plan or changes in the traffic might trigger the high-level plan to change. Humans can memorize routes that they drive frequently, but when driving to new locations, humans need to pay attention to traffic signs or use a GPS. An autonomous vehicle needs similar information to make high-level decisions at intersections.

9.2.1. High-level planning as supported in CARLA

Carla creates a compound map of the town where the ego vehicle will operate. The map contains both a 3D model of the town and its road definition. The data that defines the roads is retrieved from an OpenDRIVE file that describes the town. OpenDRIVE is an open format specification to describe the logic of road networks. The road definition contains different data structures that together define the road structures. The data structures include information about traffic signs, waypoints, lanes, and junctions. In addition to these data structures, Carla operates with four different navigational commands; left, right, straight, and follow the lane. The Carla team uses this information to create high-level plans between two locations. An illustration of the road network in Town01 is shown in Figure 9.1.

The CARLA team provides code that finds the shortest path between two waypoints in the road network. The A* search algorithm with distance heuristic is used to find the high-level plan between two waypoints in the road network. The CARLA team has implemented a class that creates a list of tuples on the form (waypoint, navigational command) from a start location to an end destination. The waypoint is a point in the middle of a lane with orientation, and the navigational command is one of the predefined commands made by the CARLA team. The sampling resolution between the tuples is set when an object of the class is created. A spawning position is a term the CARLA team uses to describe a position in the simulation where a vehicle can be initialized without creating a collision. Figure 9.2 show the possible spawning positions for a vehicle, and the generated high-level plan from a random start location to a random end location. Figure 9.2b illustrates a high-level plan that is generated between two locations in Town01. It shows the location of the waypoints in the route together with the navigational command which is illustrated with different colors. The



(a) An overview of the possible starting and ending locations in Town01. The blue dots are predefined locations where the vehicle easily can be teleported to.

(b) The yellow points correspond to a follow lane command, the turquoise points correspond to a right turn, the blue points correspond to a left turn, and the purple points correspond to straight in junction. The green point is the start location, and the red point is the end location.

Figure 9.2.: Possible spawning positions in Town01 for the ego vehicle and a generated high-level plan between two locations.

route includes more information than the (x, y) coordinates and the navigational command that are scattered in Figure 9.2, but this additional information is not necessary for the high-level planning in SafeRide.

9.2.2. High-level planning in SafeRide

The high-level planning in SafeRide is based on the high-level planning provided by CARLA described in subsection 9.2.1. A start position $\vec{s}$ and a goal position $\vec{g}$ are selected at the start of a ride. The shortest path between these positions that complies with the traffic rules is calculated with the high-level plan code provided by the CARLA team. SafeRide only saves tuples on the form (x_h, y_h, v_h, c_h) where x_h is the x-position in meters, y_h is the y-position in meters, v_h is the target speed in m/s, and $c_h \in \{\text{left}, \text{right}, \text{straight}, \text{follow lane}\}$ is the navigational command. The sample resolution s_r between these points is 1 meter, which means that the number of points in the high-level plan will depend on the length of the high-level plan.

The high-level plan in SafeRide is called *segmented high-level plan*. This high-level plan is a moving segment of the complete high-level plan provided by CARLA. SafeRide uses a moving segment of the high-level plan from CARLA to create targets for the short-term planner with imitation learning, which will be explained in subsection 9.5.2. The moving segment starts at the current location of the ego vehicle, and the length is determined by the desired velocity v_h at the point in the high-level plan closest to the ego vehicle, and the length of the short-term plan T_f in seconds described in section 9.4. The number of points n in the segment of the high-level plan is given by,

$$n = \left\lceil \frac{v_h}{s_r} \cdot (T_f + 1) \right\rceil \tag{9.1}$$

The segmented high-level plan has a variable number of points since it is used to select the best short-term plan (see section 9.7) to be the target for the short-term planner with imitation learning.

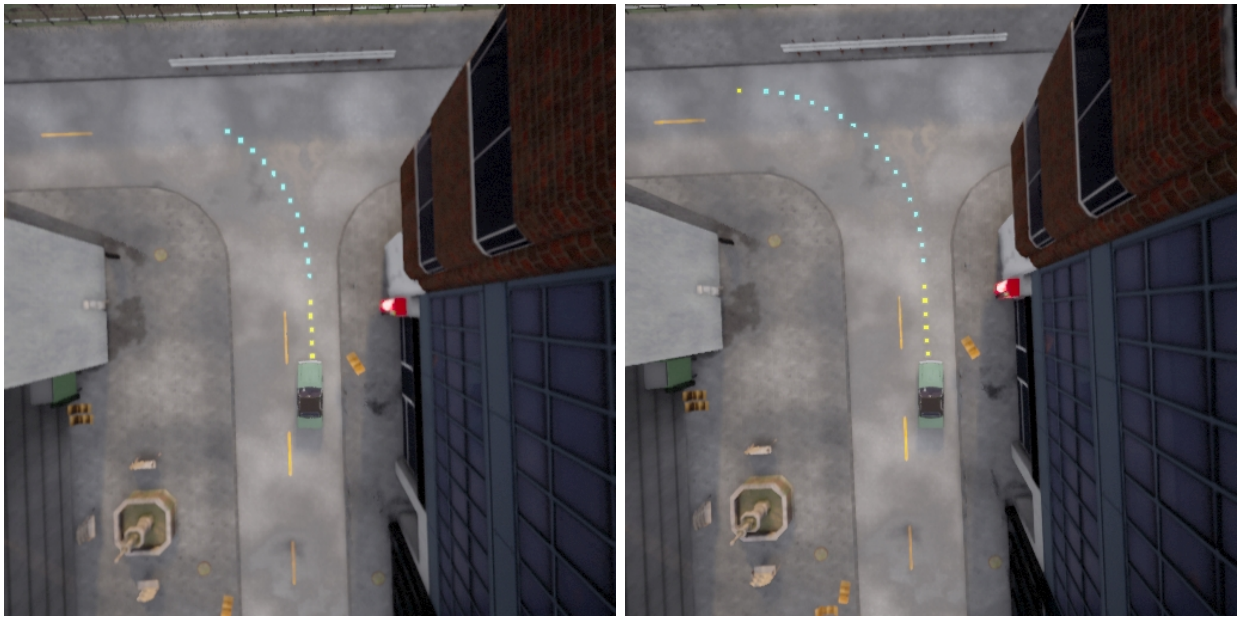


Figure 9.3.: Two segmented high-level plans that show how the desired velocity v_h affects the number of points in the segment. The ego vehicle in the figure to the left has a lower v_h than the ego vehicle in the figure to the right.

Figure 9.3 illustrates how a change in desired velocity v_h influence the number of points in the segmented high-level plan. The segment is required to be longer than the generated short-term plans, but it is also beneficial to keep it as short as possible because of the computational cost. The motivation for varying the number of points n in the segmented high-level plan will be further clarified in section 9.7.

9.3. Navigation in SafeRide

Navigational systems offer sparser data on a higher level than a segment of points, as is the case for the high-level plan. Therefore, we introduce an attraction point that will be used to generate the navigational instructions for the SafeRide system. The attraction point is the first point in the high-level plan where the navigational command changes. The navigational instruction consists of three different values. The first value is the navigation command before the attraction point. This is a categorical value which informs the vehicle about what command that need to be performed until this point. The second values in the navigation instruction is the next navigational command. The last value in the navigation instruction is the distance to the attraction point, d_a. This is calculated by summing the distance between the n subsequent points x_h before the attraction point,

$$d_a = \sum_{i=0}^{k} \|x_h^{(i+1)} - x_h^{(i)}\|. \tag{9.2}$$

Altogether, these three values correspond to a common GPS-based navigation system. Navigational commands in such systems could for instance be; follow lane for 250 meters, then take a right turn. The navigational instructions in SafeRide contain the same information that is given in the example. Figure 9.4 illustrates two different attraction points. Figure 9.4a shows an attraction point that is processed to the navigational instruction follow lane for 10 meters, then take left. Figure 9.4b

(a) Attraction point that is translated to the navigation instruction follow lane 10 meters, then take left.

(b) Attraction point that is translated to the navigation instruction take left 15 meters, then follow lane.

Figure 9.4.: Example of two different points in the high-level plan that are used for navigation. The color of the attraction point in the figure to the left illustrates that the navigational command is follow lane until this point. The color of the attraction point in the figure to the right illustrates that the navigational command is left until this point.

shows an attraction point that is processed to the navigational instruction take left for 15 meters, then follow lane. The attraction point will incentivize the autonomous vehicle to make the correct decision at intersections. However, the autonomous vehicle is not forced at any cost to take the proposed decision. The navigational instruction is less strict than the high-level plan.

9.4. Control action sequence generation

In this thesis we define a *control action sequence* to be a sequence of steering actions z and throttle actions r. The control action sequence is used by the complete motion model described in chapter 8 to predict possible short-term plans. Steering actions cannot change arbitrarily fast, therefore it takes some time to turn the steering wheel. The same argument holds for the pressing the throttle pedal. This means that it is not possible to change the steering action z or throttle action r from -1 to 1 during a short time interval. Therefore, the steering and throttle actions need to be continuous functions over time. We propose a way to create feasible short-term plans based on the control action sequences that are piecewise linear over time. The short-term plans are executed by the *short-term plan executor* module described in chapter 10. Next, the component that creates the control action sequences is explained.

9.4.1. Driving action generator

The inputs to the complete motion model designed in chapter 8 are generated by a *driving action generator (DAG)*. The DAG outputs control action sequences from some given parameters. Figure 9.5 illustrates how the motion model transforms the control action sequences from the DAG to a short-term plan. A positive value of the steering input z correspond to a left turn, as seen in

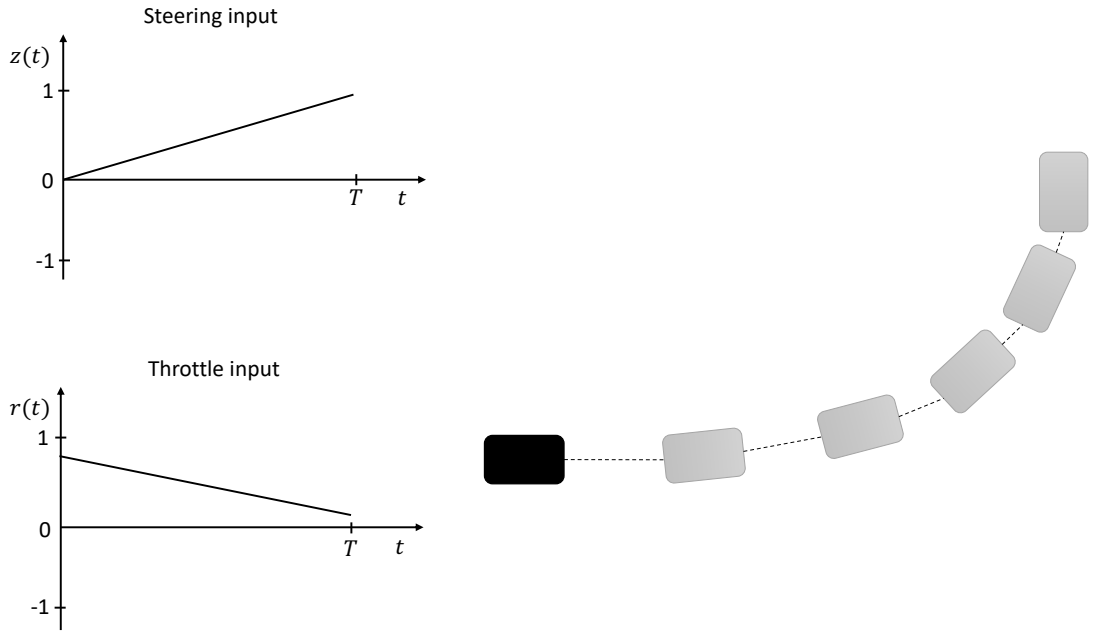


Figure 9.5.: Example of possible control action sequences for the steering input z and the throttle input r, and the corresponding short-term plan.

Figure 9.5. The curvature of the short-term plan increases since the steering input is increasing as time proceeds. This DAG outputs two linear functions over time for a given time interval T. Thus they can be represented by two parameters a and b,

$$\begin{aligned} z(t - t_0) &= z(t_0) + a \cdot t \\ r(t - t_0) &= r(t_0) + b \cdot t \end{aligned} \tag{9.3}$$

These control action sequences are continuous over time, which means that $z(t_0 + T)$ and $r(t_0 + T)$ are automatically also the start values of the next period. The input parameters to the DAG are the desired values $z(t_0 + T)$ and $r(t_0 + T)$ at the end of the interval. This way, it is ensured that $z(t)$ and $r(t)$ stay in well-defined value intervals. In other words, the DAG is just producing a smooth course of the action signals. The motion model converts the control action sequence into a prediction of the temporal evolution of the vehicle position $\mathbf{x}(t)$, the vehicle speed $v(t)$, and the orientation $\theta(t)$. Figure 9.6 shows an overview of how the DAG and the motion model are connected. The generated short-term plan (see right part of Figure 9.5 for an example) can then be checked for collisions and evaluated for different criteria.

If non-smooth control action sequences are desirable, we can use more parameters. For example if we want two curve segments $z(t_0 + T/2)$, $r(t_0 + T/2)$, $z(t_0 + T)$, and $r(t_0 + T)$ is selected. The input values to the DAG is named *local plan parameters*. In the following sections, different ways to create these local plan parameters will be presented.

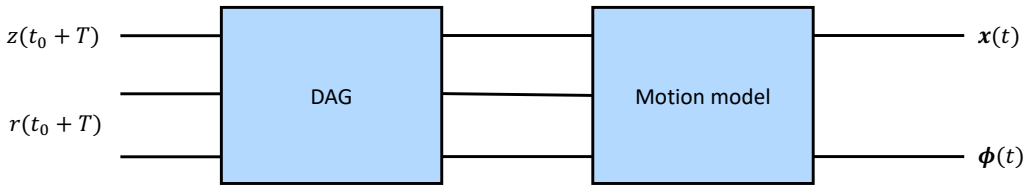


Figure 9.6.: The architecture of a short-term plan generation system. The local plan parameters describing the steering input z and the throttle input r some time T into the future are inputs to the dynamic action generator (DAG). The DAG outputs control action sequences, which a motion model transforms to a short-term plan.

9.5. Different proposals for short-term planners

In this section, different *short-term planners* are proposed. The goal of a short term planner is to produce local plan parameters which result in feasible, safe, and comfort short-term plans that leads the ego vehicle closer to the goal location. All the short-term planners share one structural element, they use local plan parameters to hallucinate short-term plans. The available information that the short-term planners can use to generate local plan parameters is defined as an *environment state*. The environment state consists of the dynamic Circogram, the potential field, the navigational instruction, and the dynamic vehicle state (position, velocity, orientation, and change in orientation).

The short-term planners are presented one after one with increasing complexity. The first short-term planner is based on random sampling, which result in a simple but inefficient short-term planner. A short-term planner without information about the situation is only an initial experiment, and the short-term planner must have some input from the current driving situation. The second short-term planner uses imitation learning to create a policy network which takes environment states as inputs and outputs local plan parameters. This short-term planner learns from prerecorded driving, which might result in dataset bias and poor generalization. The last short-term planner improves this by using reinforcement learning to allow exploration.

9.5.1. Short-term planner with random sampling

The simplest way to create the local plan parameters described in section 9.4 is to select them by random sampling. This way of creating local plan parameters is named *short-term planner with random sampling (STP-RS)*. This way to create the local plan parameters is only meant as an initial experiment due to the low efficiency of evaluating many short-term plans. The efficiency loss results from the fact that the sampling is not controlled by the current driving situation. The short-term planner does not know about the current steering needs and will therefore most often produce plans which are not feasible since they would lead to collisions. However, it is possible to randomly draw the local plan parameters from a specified distribution. The probability is high that one of the resulting short-term plans are good if many short-term plans are created. It is explained later why this is not a good approach, and better approaches will be presented later in this chapter.

Finite state machine with random local plan parameters

The control action sequence generation in section 9.4 requires a set of local plan parameters to output short-term plans. One way to create these parameters is by random sampling. In this section, we design two different ways of sampling the parameters. These two approaches to create random parameters are included as states in a *deterministic finite state machine (DFSM)* that decides which way to random sample the parameters.

Driving consists primarily of two different types of plans. Either the driver follows the high-level plan, or an event happens, and the driver needs to do a maneuver. Therefore, we define a steady-state and an event state. These two states sample the random parameters in different ways.

In the steady-state, we assume that the steering and throttle actions do not change that much. At t_0 the actuators are on the current state, and the next parameter will be this state plus some small deviation. This is the same as using the previous parameter as the mean in a normal distribution, and setting some small standard deviations σ_z and σ_r. Hence, in this state we sample the next local plan parameters from a normal distribution where the previous local plan parameter is the mean as,

$$\begin{aligned} z(t_{i+1}) &\sim \mathcal{N}(z(t_i), \sigma_z) \\ r(t_{i+1}) &\sim \mathcal{N}(r(t_i), \sigma_r) \end{aligned} \tag{9.4}$$

If all the generated short-term plans from the steady state parameter generation lead to a collision, then we switch to the event state. In the event state, we do not make any assumptions about the parameters. Therefore, we sample the local plan parameters from a uniform distribution over the whole possible interval.

$$\begin{aligned} z(t_{i+1}) &\sim \mathcal{U}(-1, 1) \\ r(t_{i+1}) &\sim \mathcal{U}(-1, 1) \end{aligned} \tag{9.5}$$

Fine-tuning of random local plan parameters

Generating random local plan parameters from a uniform distribution results in a small set of feasible short-term plans. This means that the system can select between few possible short-term plans. To create a larger set of possible short-term plans, we propose to create the random parameters in two steps. The first step is to generate random parameters from a uniform distribution as in Equation (9.5). The best of the possible parameters, $z_b(t_{i+1})$ and $r_b(t_{i+1})$, is then fine-tuned. The next step is to draw parameters from a normal distribution with small standard deviations σ_z and σ_r, and mean $z_b(t_{i+1})$ and $r_b(t_{i+1})$,

$$\begin{aligned} z(t_i) &\sim \mathcal{N}(z_b(t_i), \sigma_z) \\ r(t_i) &\sim \mathcal{N}(r_b(t_i), \sigma_r) \end{aligned} \tag{9.6}$$

The fine-tuning creates a more extensive set of suitable local plan parameters and will improve the planning. This improvement of the parameters from this two-step process comes with an additional computational cost. The random generation of local plan parameters is not an efficient approach and this approach is only an initial experiment. Nevertheless, this process will be used to create target parameters for the short-term planner with imitation learning.

9.5.2. Short-term planner with imitation learning

A more efficient approach is to use learning to create the local plan parameters. Imitation learning has previously shown remarkable performance for autonomous vehicles in CARLA [Chen et al. 2020]. Imitation learning is performed by collecting data from an expert driver or an autopilot and then using this data in a supervised learning manner. The data which needs to be collected is the so-called input to the neural network, and the corresponding so-called target. The input is the environment state, and is a combination of a dynamic Circogram, a potential field, a dynamic vehicle state, and a navigational instruction. The navigation commands in the navigation instruction are one hot encoded. The corresponding target is the local plan parameters. The *short-term planner with imitation learning (STP-IL)* learns a mapping from an environment state to local plan parameters. The environment states can be saved at every frame, but the target values in the SafeRide system are harder to obtain. It is necessary to preprocess data from the simulation to create the local plan parameters, or use the STP-RS described in subsection 9.5.1. An expert driver does not perform a linear change in steering and throttle every T, which is assumed by the DAG described in section 9.4. The violation of the assumptions in section 9.4 means that it is necessary to preprocess the expert actions to create appropriate local plan parameter targets. The targets can be created by taking the average of all the actions in the interval, or simply using the action T seconds into the future as the action. This way of creating the targets is a considerable simplification and assumes that the expert selects smooth actions. Then the target y at time t can be created by,

$$y_t = \frac{1}{n}\sum_{i}^{n} \mathbf{a}_{t+\Delta t*i} \qquad y_t = \mathbf{a}_T \tag{9.7}$$

where y_t is the target at time t, a_t is the actuator state at t, $n = \frac{T}{\Delta t}$ is the number of timesteps to T. Another approach is to carefully select the data where a linear regression of the actions for the next T seconds have residuals below a threshold. This approach will create better targets, but some driving situations will never yield results where the residuals are below some threshold. Targets from only simple driving scenarios will result in an agent who only learns simple maneuvers where the actions are almost linearly.

The short-term plans that are created from STP-RS as described in section 9.5.1, perform very well in initial experiments. This process is therefore used to create targets for STP-IL. The creation of the targets is done by generating 200 random short-term plans with the two-step process explained in section 9.5.1. This means that 400 short-term plans are created and evaluated for every target that is created. The random generation yields good short-term plans, but it is not efficient enough to use in the final SafeRide system. The best short-term plan is selected based on section 9.7, which will be described later. The local plan parameters that result in the best short-term plan will become the local plan parameters for the specific environment state.

The produced local plan parameters from the STP-IL are assumed to be means of normally distributed random variables. Each of the means are connected to small standard deviations, σ_r and σ_z. The produced means and the standard deviations are used to generate several feasible short-term plans. The generation of the short-term plans is completed by the DAG and the complete motion model. This process is done to improve the robustness of the system. Hence it is possible to generate several short-term plans from one output from the STP-IL.

9.5.3. Short-term planner with reinforcement learning

Imitation learning is criticized for dataset bias and overfitting [Codevilla et al. 2019]. Reinforcement learning is an approach that overcomes these limitations by exploring unknown environment states. The reinforcement learning algorithm is used to create local plan parameters for the vehicle in the same way as the imitation learning described in the previous section. This way of learning the local plan parameters is named *short-term planner with reinforcement learning (STP-RL)*. The environment states consists of the same information as in the imitation learning setup. The output is the local plan parameters. This means that the STP-RL should output the steering input and the throttle/brake input some seconds into the future. The produced local plan parameters from the STP-RL are assumed to be means of normally distributed random variables. Each of the means is connected to standard deviations, σ_r and σ_z. The produced means and the standard deviations are used to generate several feasible short-term plans. This process is the same as for the STP-IL.

The reward function should reflect what many safe, comfort, efficient and feasible short-term plans are generated. One way to design the reward function is use the amount of feasible short-term plans times the total short-term plan score. The short-term plan score ρ is explained in section 9.7, and a high value is beneficial. The STP-RL receives a negative reward if the vehicle crashes, or drives into the opposite driving lane. Although this might result in an autonomous vehicle that never overtakes other vehicles. The reward is therefore defined as,

$$R(s,a) = \begin{cases} -10 & \text{if crash} \\ -1 & \text{if all short-term plans lead to crash} \\ -1 & \text{if outside driving lane} \\ \kappa \cdot \rho & \text{else} \end{cases} \tag{9.8}$$

where κ is the proportion of the short-term plans that are feasible, and ρ is the short-term plan score defined in section 9.7. The STP-RL will learn to output many short-term plans with good scores, which is the goal.

Speeding up learning

Reinforcement learning agents often take in general long time to learn a policy or a value function that performs well. One can use imitation learning to speed up the training to overcome this limitation. The speedup is done by initializing the reinforcement learning policy with the trained weights from imitation learning. This approach uses the same policy network that is proposed for STP-IL, shown in Figure 9.7.

9.6. Policy network for the learning-based short-term planners

The architecture of the neural network is illustrated in Figure 9.7. The policy network that maps environment states to local plan parameters consists of two feature extractors and a dense feed forward neural network head. The two feature extractors take the intermediate environment representations as inputs. The dense feed forward neural network takes the output from the feature extractors, the dynamic vehicle state, and the navigation instruction as inputs. The policy network yields local plan parameters which represent the mean of a stochastic variable. These three components will be described sequentially in the next sections.

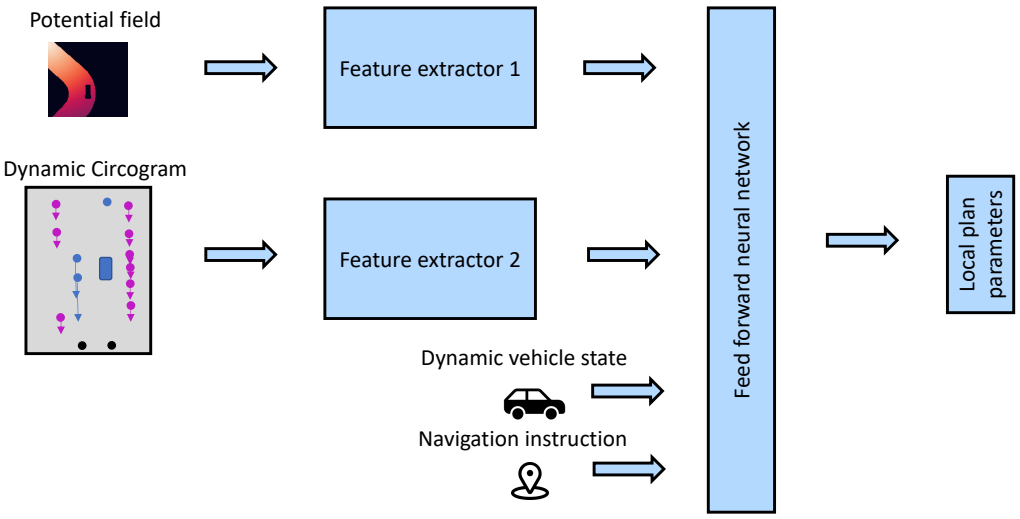


Figure 9.7.: Policy network architecture that takes the potential field, the dynamic Circogram, the dynamic vehicle state, and the navigational instruction as input, and outputs local plan parameters. The neural network consists of two feature extractor, and one feed forward neural network head.

9.6.1. Feature extraction of the potential field

As explained in section 7.2, all the values in the potential field that do not correspond to free space have the same big value. All the values in the free space can be deduced from the neighbors. This means that the potential field contains redundant data, which means that it should be possible to compress the potential field to a latent vector $\mathbf{h}$. This vector should contain all the information relevant in the potential field. One way to compress images to latent vectors is to use an autoencoder. An autoencoder is trained on a big dataset with potential fields. The potential fields contain some high values, and are therefore scaled. All the values are divided by 512 because the recolored BEV, which the potential field is created from, has the shape $512x512$. The creation of potential fields is efficient, which makes it possible to create a huge dataset with such images. Figure 9.8 shows the autoencoder structure that SafeRide uses. The encoder consists of three 2-dimensional convolutional layers, and one dense layer, while the decoder consists of one dense layer and three 2-dimensional transposed convolutional layers. All the activation functions are ReLu-functions. The input to the encoder is the potential field which is a $128x128$ matrix with only positive values. The potential field on the left side of the figure is an example of an original image of the potential field, and the potential field on the right side of the figure is the corresponded recreated potential field. The brighter color outside the free space is due to some high values that influences the color map. Figure F.1 shows several examples of original potential fields and the recreated potential fields from the decoder part of the autoencoder. The autoencoder is able to preserve the temporary goal pose and the location of the free space. These are the important information from the potential field, which means that the autoencoder is able to compress the potential field to 128 features that represent this information.

The autoencoder is trained by itself on a predefined dataset of potential fields. The potential fields are generated from driving with an autopilot. Only the trained encoder is used in the SafeRide

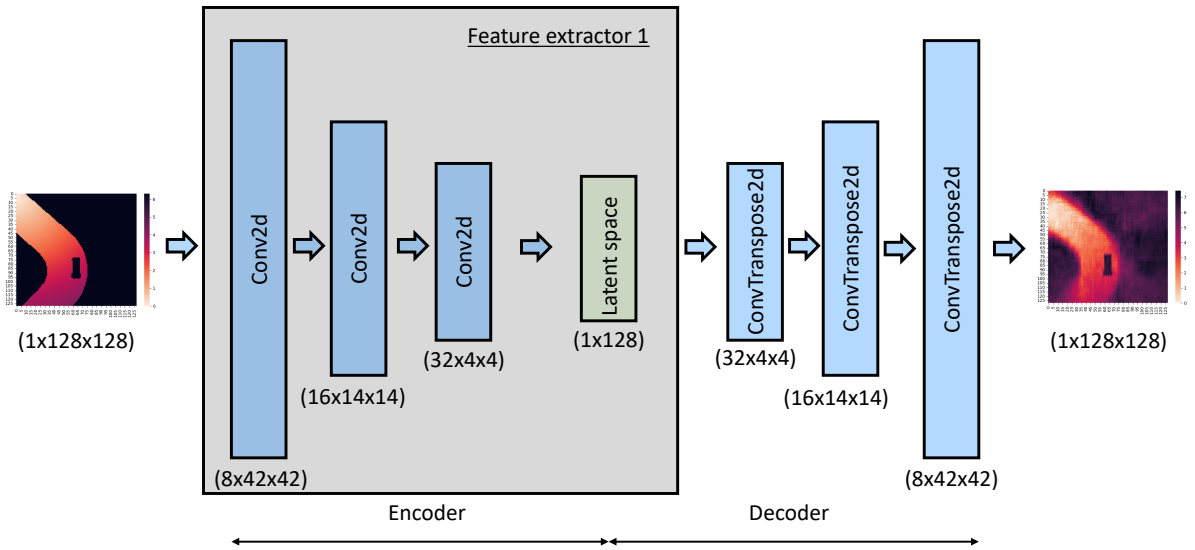


Figure 9.8.: Convolutional autoencoder structure for the feature extractor of the potential field.

system. The weights in the encoder called feature extractor 1 are frozen after the training. Hence this feature extractor is only used to compress the potential field to a latent vector which contain information that is used to find good local plan parameters.

9.6.2. Feature extraction of the dynamic Circogram

The second part of the environment state is the dynamic Circogram. The dynamic Circogram is a sparse representation extracted from the recolored BEV and the motion of all the dynamic objects in the recolored BEV. It is not possible to further compress this data if the number of simulated rays are kept constant, and hence it is not reasonable to build an autoencoder for this part of the policy network. However, it is appropriate to use convulutional layers due to the spatial relationship between subsequent points in the dynamic Circogram.

In this architecture, it is assumed that the dynamic Circogram contains 100 points. This feature extractor is build up of one-dimensional convolutional and max-pooling layers. The design is inspired by the VGG-16 architecture by Liu and Deng 2015. Two convolutional layers with the same number of channels are followed by a max-pooling layer. Figure 9.9 shows the structure of the feature extractor of the dynamic Circogram. The feature extractor of the dynamic Circogram is not pretrained.

9.6.3. Feed forward neural network of concatenated features

The last part of the policy network uses the extracted features, the dynamic vehicle state, and the navigation instruction as inputs and outputs the mean of local plan parameters. The features from the feature extractors described in subsection 9.6.1 and subsection 9.6.2 are concatenated with the dynamic vehicle state, and the navigation instruction. The concatenated data is passed through two dense layers, which results in mean values of the local plan parameters.

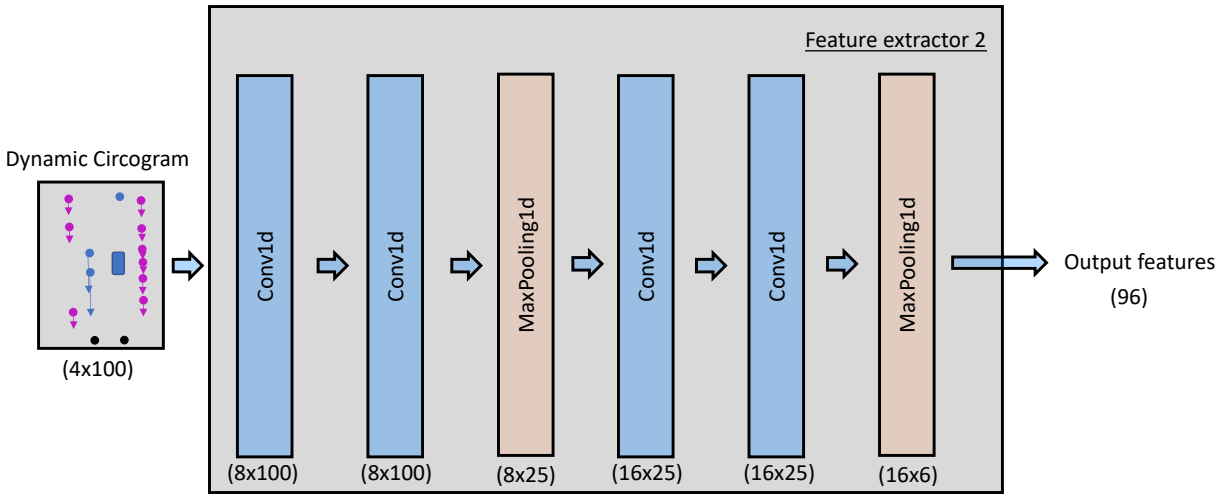


Figure 9.9.: Convolutional neural network structure for the feature extractor of the dynamic Circogram. The convolutional neural network consists of convolutional and max-pooling layers.

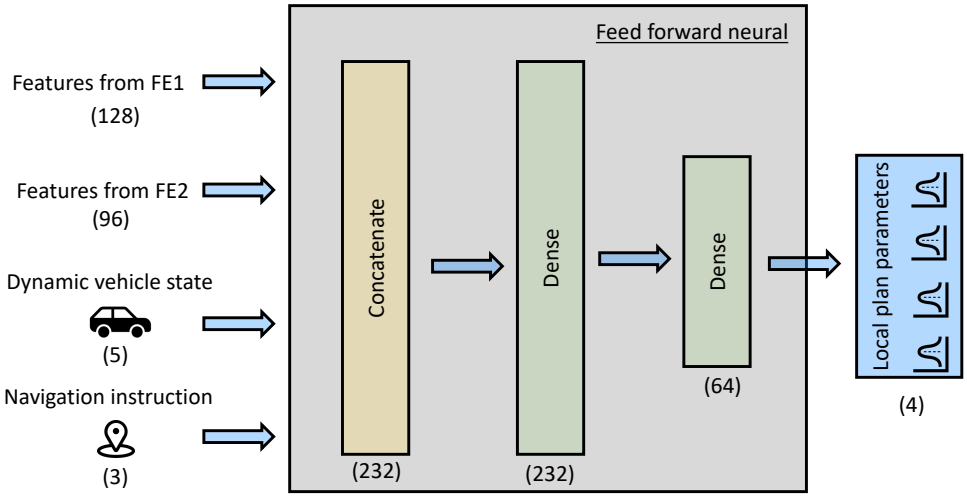


Figure 9.10.: Feed forward neural network architecture of the head of the policy. The feed forward neural network consists of a concatenation of the features and two dense layers.

In this specific architecture, the block only yields four outputs which correspond to the mean of the local plan parameters that the DAG uses. There are two parameters for steering and two parameters for throttle. The output can be extended in two different ways. First, it is possible to increase the number of outputs to represent several short-term plan segments. This will increase the complexity of the short-term plans that are generated. The other extension is to also output the standard deviation to the mean parameters. The reasoning behind this is that highway driving is more predictable than urban driving and therefore needs a lower standard deviation in the short-term plan generation. Nevertheless, in this thesis we only yield the mean parameters, and assume that that the standard deviation is a small constant number. The decision to treat the outputs as normal random variables makes it possible to quickly generate a small set of local plan parameters. Providing a set of good local plan parameters will increase the robustness of the autonomous vehicle, but it will be necessary to have a procedure of finding the best local plan parameters in the set.

9.7. Short-term plan selection

It is necessary to select the best short-term plan if the short-term planner proposes several short-term plans. This selection process needs to be completed in two steps. First, one needs to remove short-term plans that lead to a collision and do not obey the traffic rules. Then one needs to find the best short-term plan of the remaining based on optimization with some criteria.

9.7.1. Hard constraints to avoid collision and obey speed limits

The first step in the short-term plan selection procedure is to remove the short-term plans that lead to a collision. The short-term plans describe how the center of the vehicle will move, but it is necessary to use the rigid body of the vehicle at each point. The rigid body of the vehicle is approximated with four circles, which makes it fast to check for collisions. For each point in the short-term plan, four points (x_f, y_f) are distributed along the longitudinal axis of the vehicle. These points are located a distance r_i away from the center of gravity of the vehicle in the vehicle coordinate frame. Two of the points are placed in front of the center of gravity, and the other two points are placed behind the center of gravity. The center of the circles are calculated by,

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = \begin{bmatrix} x_c + r_i \cdot \cos(\phi) \\ y_c + r_i \cdot \sin(\phi) \end{bmatrix} \tag{9.9}$$

This results in four centers of circles which are used to check if the short-term plan lead to a collision. This is illustrated in Figure 9.11. The collision check is done by calculating the distance from the hitpoints in the Circogram to the center of the circles, and checking if it is bigger or smaller than the radius of the circles. If one point from the Circogram is inside one of the circles, then the short-term plan will lead to a collision. To further improve safety, a safety margin is included. This is done by enforcing that the distance between the hitpoints from the Circogram and the circles is above some threshold. In this thesis, we use a safety distance of 0.1m.

The second hard constraint is the speed limit. The ego vehicle is not allowed to drive faster than the speed limit, and therefore all the short-term plans that include a speed higher than the speed limit are removed. This is checked by finding the closest point in the high-level plan for each point in the possible short-term plan and checking the speed in the short-term plan against the corresponding speed in the high-level plan. Given a point (x_c, y_c, v_c, ϕ_c) in a possible short-term plan and a high-level segment with n points $(x_h^{(i)}, y_h^{(i)}, v_h^{(i)}, c_h^{(i)})$ where $i = 1, \ldots, n$. Then the closest point in the

Figure 9.11.: Illustration of four circles that approximate the outer surface of the ego vehicle.

high-level plan k is found by,

$$k = \arg\min_i \sqrt{(x_h^{(i)} - x_c)^2 + (y_h^{(i)} - y_c)^2} \tag{9.10}$$

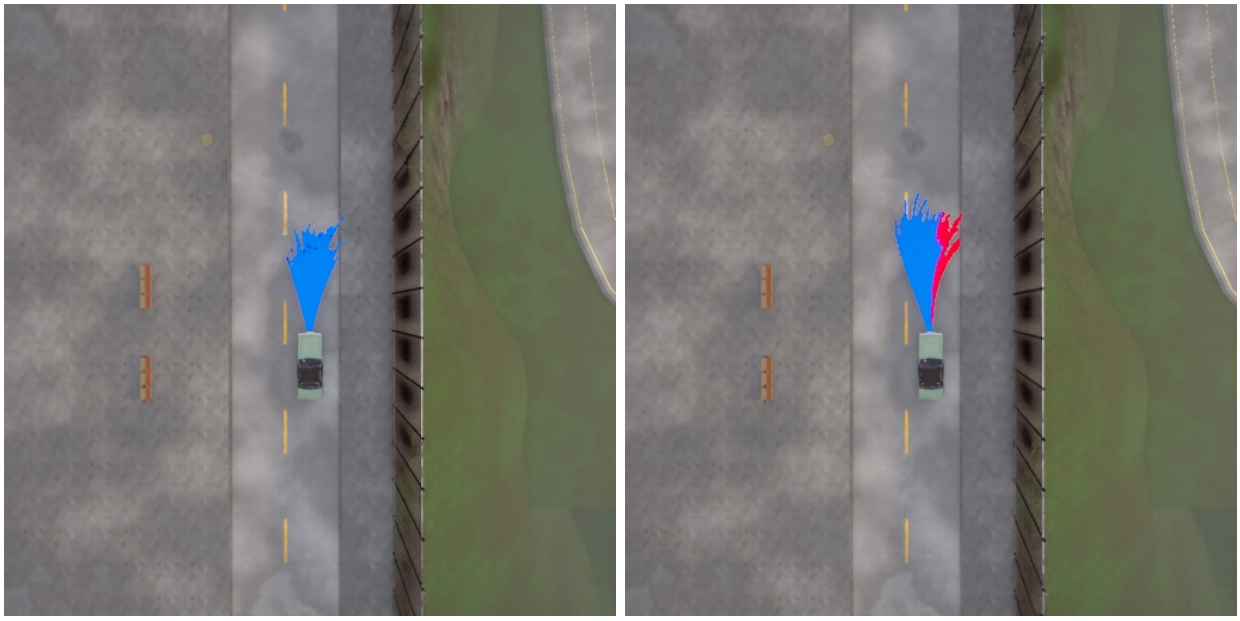
The speed v_c is checked against the speed $v_h^{(k)}$. If $v_c > v_h^{(k)}$, then the short-term plan is removed from the set of possible short-term plans. There might be beneficial to let the autonomous vehicle drive faster than the speed limit in some situations to avoid dangerous situations. This aspect is not considered in the SafeRide system. Figure 9.12 shows the process of applying the hard constrains on a set of short-term plans. All the generated short-term plans are shown in Figure 9.12a. The short-term plans that violate the speed limit or lead to a collision is detected. These short-term plans are colored red in Figure 9.12.

9.7.2. Soft plan selection criteria

The set of short-term plans that do not violate any of the hard constraints is further evaluated. The best short-term plan needs to be selected for the current pose of the car from the set of possible short-term plans. This short-term plan should lead to the expected behavior described in section 9.1 in the beginning of this chapter. The best short-term plan is the one that leads the car towards the goal state and at the same time keeps the ego vehicle comfortable and safe. Criteria that consider the risk, the comfort, the target speed, and the high level plan are presented in the next sections.

Risk criteria

Risk is one of the objectives that is used for selecting the best short-term plan. It is necessary to define what a safe short-term plan is with mathematical notation. One way to measure the risk is to use the distance to the surrounding objects together with the relative motion vector. A safe short-term plan should keep a safe distance to other objects, especially other objects that move towards the ego vehicle.



(a) 200 generated short-term plans from STP-R. These short-term plans are generated with the two-step process explained in section 9.5.1.

(b) The result after applying the hard constraints. The red short-term plans violates one of the hard constraints. The blue short-term plans fulfill the hard constraints, which means that they do not violate the speed limit or lead to a collision.

Figure 9.12.: The process of how the hard constraints in the short-term plan selection reduce the set of feasible short-term plans.

Time to collision (TTC) is a widely used time based risk indicator that calculates the time to a possible collision for the ego vehicle. The TTC value at an instant time is defined as the time for the ego vehicle and an other object to collide if they continue at their current speed and on the same short-term plan. A negative value of TTC means that the vehicles are moving apart from each other. The Dynamic Circogram presented in section 7.1 include the necessary information to calculate the TTC, which can be used as a risk measure.

The Dynamic Circogram contains the relative velocity $\mathbf{v}_{i/s}$ between the ego vehicle and each hitpoint from the simulated rays explained in section 7.1. This is used together with the vector from each surface point $\mathbf{x}_o$ on the vehicle to the hitpoints $\mathbf{x}_d$. The vector between these points is defined as $\ell = \mathbf{x}_d - \mathbf{x}_o$, which is the vector between the outer surface of the vehicle and the hitpoint. A risk criteria τ is defined to be the projection of the relative velocity $\mathbf{v}_{i/s}$ onto the vector ℓ multiplied by the norm of ℓ. τ for a hitpoint i is then computed by,

$$\tau_i = |\ell| \cdot \mathbf{v}_{i/s} \cdot \ell \tag{9.11}$$

By analyzing the projection and $|\ell|$ on its own, it is possible to understand which values of τ_i that are risky. This is illustrated in Figure 9.13. Case 2 in the figure has a negative τ_i because of the negative value of the projection. This means that the ego vehicle is driving away from the hitpoint, and the situation is not considered as risky. Case 1 and case 3 has equal positive value of the projection, but the difference to the hitpoint differ. The ego vehicle is closer to the hitpoint in case 1 compared to case 2. Therefore, case 1 is more risky since it has a smaller $|\ell|$. If $\mathbf{v}_{i/s}$ and ℓ are pointing is the same direction, then the projection will be close to zero. On the other hand, the projection is negative if they are pointing in opposite direction. This means that the most risky short-term plan has a

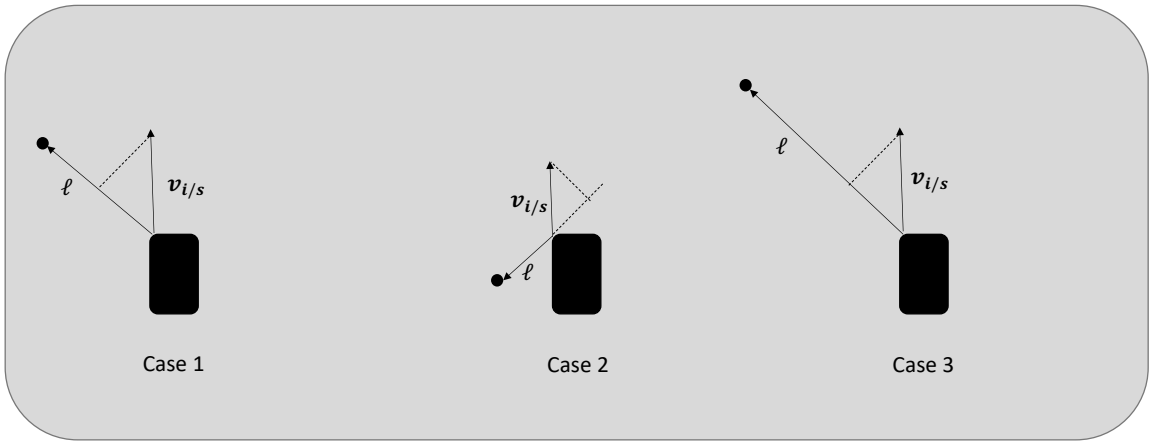


Figure 9.13.: Illustration of how the relative velocity vector $\mathbf{v}_{i/s}$ and the distance vector $|\ell|$ on the risk measure τ_i. Three different cases show how the length of the distance vector and the projection of the relative velocity onto the distance vector.

projection that is positive and close to zero. ℓ is used to scale the projection such that the distance between the ego vehicle and the surrounding objects is considered. To summarize, a surrounding object is considered to be risky if the relative velocity between it and the ego vehicle is pointing in the same direction as the vector between them and that they are close.

The TTC τ_i is calculated between each pair of outer surface points of the vehicle and the hitpoints. A negative τ_i means that the ego vehicle is not going to hit that object. For positive values, a τ_i close to 0 is risky. Therefore, we select the minimum of the positive values τ_i^+ to represent the risk,

$$\tau = \min(\tau_i^+) \tag{9.12}$$

It is necessary to assess the risk along the whole short-term plan. This is done by calculating the outer surface of the ego vehicle at each timestep t along the short-term plan, and summing the τ_t for all t. The risk criterion based on the TTC ρ_1 is defined as,

$$\rho_1 = \sum_{t=0}^{T} \tau_t \tag{9.13}$$

Comfort criteria

The comfort of the passengers is also an important measure of how good the short-term plan is. Change in the lateral acceleration is perceived as unpleasant for passengers [Hayati et al. 2020].

Jerk is the third derivative of the position with respect to time, and is defined as,

$$\mathbf{j}(t) = \frac{d^3\mathbf{r}(t)}{dt^3} \tag{9.14}$$

The lateral jerk is the x-component of the jerk in the ego vehicle coordinate frame. This will be caused by centripetal acceleration when entering a curved path. A short-term plan with lower jerk will increase the comfort for the passengers in the ego vehicle. For a short-term plan $\mathbf{r}(t)$, the total jerk will be the integral of the squared jerk from the start of the short-term plan t_0 to the end of the short-term plan t_1,

$$H(\mathbf{r}(t)) = \int_{t_0}^{t_1} \dddot{\mathbf{r}}_{\mathbf{x}}(t)^2 dt, \tag{9.15}$$

where $\mathbf{r}_{\mathbf{x}}(t)$ is the lateral component of the short-term plan. The total squared jerk from (9.16) can be discretized as,

$$\rho_2 = \sqrt{H[\mathbf{r}[t_i]]} = \sqrt{\sum_{t_0}^{t_N} \left(\frac{a_x[t_i] - a_x[t_{i-1}]}{\Delta t} \right)^2}, \tag{9.16}$$

where $a_x[t_i] = \frac{v_x[t_i] - v_x[t_{i-1}]}{\Delta t}$ and $v_x[t_i]$ is the velocity at time t_i in x-direction from the generated short-term plan, $\Delta t = t_i - t_{i-1}$, and t_0 and t_N are respectively the first and the last timestep in the short-term plan.

Target speed criteria

It is also desirable that the autonomous vehicle drives with a speed that is close to the speed limit when it is possible. Therefore, the deviation between the speed along the short-term plan and the closest corresponding point on the high-level plan is calculated. The index of the closest point on the high-level plan is k, and calculated with Equation (9.10) for each point on the possible short-term plan. It is beneficial that ρ_2 is as small as possible.

$$\rho_3 = \sqrt{\sum_{t=1}^{T} \left(v_c^{(t)} - v_h^{(k)} \right)^2} \tag{9.17}$$

The aim is to have a short-term plan where this criteria is close to zero.

High-level plan criteria

The autonomous vehicle also need a criterion that makes the autonomous vehicle follow the high-level plan if it is possible. We use a soft constraint for the high-level plan because it should be possible to deviate from the high-level plan to avoid dangerous situations. For example, the autonomous vehicle should be able to deviate from the high-level plan in an intersection if a pedestrian suddenly runs into the lane that the high-level plan suggests. Nevertheless, we want all the points in the short-term plan to be as close as possible to the nearest point in the high-level plan. Again, the index of the closest point on the high-level plan is k, and calculated with Equation (9.10) for each point on the possible short-term plan. The square root of the sum of the squared distances is used as

a measure of the deviation between the short-term plan and the high-level plan, and it is calculated by,

$$\rho_4 = \sqrt{\sum_{t=1}^{T} \left(x_c^{(t)} - x_h^{(k)}\right)^2 + \left(y_c^{(t)} - y_h^{(k)}\right)^2} \tag{9.18}$$

This criteria is based on the high level plan segment, described in subsection 9.2.2. This information is not realistic to obtain from a navigational system, and therefore it will only be used to create targets for the imitation learning. A more realistic criteria is based on the second high level plan described in subsection 9.2.2, which is the attraction point. This criteria will be based on the distance between the end point x of a short-term plan and the next attraction point p,

$$\rho_5 = \|x - p\| \tag{9.19}$$

9.7.3. Short-term plan optimization

The selection of the best short-term plan is based on one criteria from all the categories in subsection 9.7.2. This is a non-trivial task, since increasing the score from one criteria might reduce the score of other criteria. Each criteria is treated as an objective, and the problem is recognized as a multi objective optimization problem [Miettinen 1998]. This is a wide group of optimization problems that depend on the individual objectives. One group of solution methods is called a priori solutions. These solution methods require that sufficient preference information is expressed before the optimization process [Hwang and Masud 1979]. It is defined that either a lower or higher value is preferred for all the individual criteria in the SafeRide system, and hence the requirement for using a priori methods is fulfilled.

One simple a priori method is to transform the multi objective optimization to a single objective optimization method. This means that we are using a function g that takes the individual criteria as inputs to form a new optimization formulation,

$$\max_{x \in X} g(\rho_1(x), \rho_2(x), \rho_3(x), \rho_5(x))^1. \tag{9.20}$$

Hwang and Masud 1979 propose scalarizing as a simple a priori method. Scalarizing require that each individual criteria is multiplied by a weight w_i where $i = 1, \ldots, 5$. The scaled criteria are summed together, and the sum is optimized as a single objective optimization.

The proposed criteria are in different ranges. This might lead to problems since one criteria might dominate the other criteria. The weights in the scalarizing is meant to handle this issue, but it is hard to select the correct weights. Normalization is used to avoid some criteria to dominate others and to make the selection of appropriate weights simpler. Normalization is performed to avoid the different objectives to have different scales. After the normalization the scores for each criteria has a mean of 0 and a standard deviation of 1. The normalized value for ρ_i is calculated by subtracting the mean $\bar{\rho}_i$ and dividing by the standard deviation σ_{ρ_i},

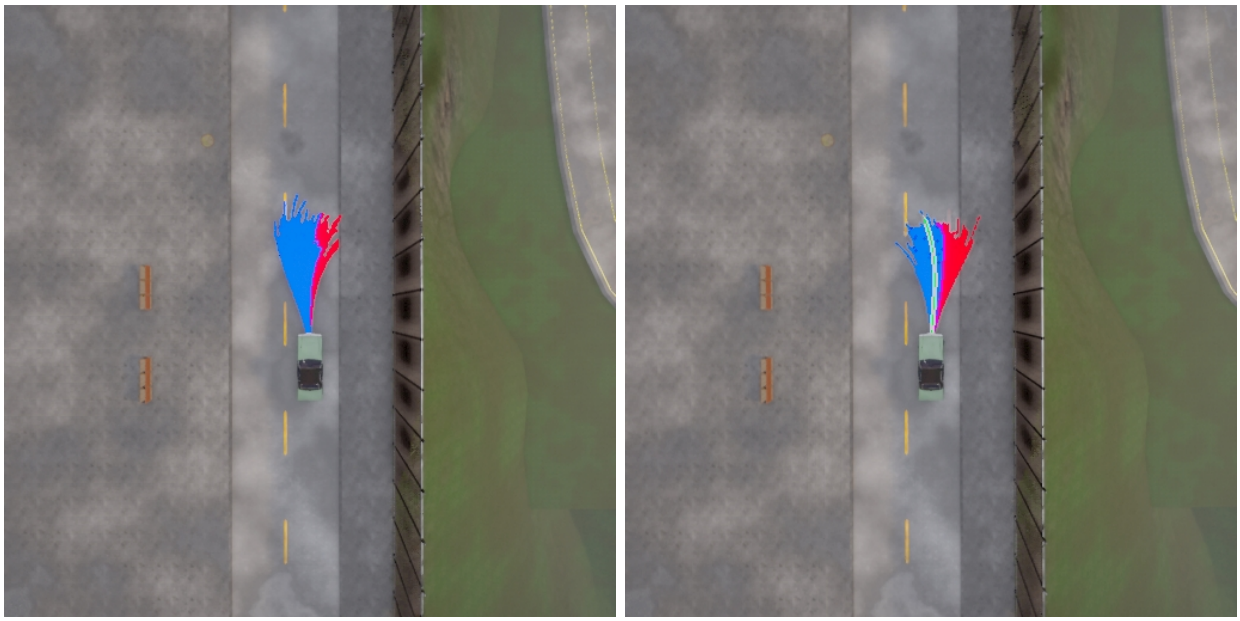
$$\rho_i' = \frac{\rho_i - \bar{\rho}_i}{\sigma_{\rho_i}} \tag{9.21}$$

[1] $\rho_4(x)$ is not considered here since it is only used to create local plan parameter targets for the STP-IL.

Since all the criteria are in the same interval after the normalization, g is defined to add the values that are preferred to be large and subtract the criteria that we want to be small. This means that the multi objective optimization problem is transformed to the following single objective optimization problem,

$$\max_{x \in X} w_1 \rho_1'(x) - w_2 \rho_2'(x) - w_3 \rho_3'(x) - w_5 \rho_5'(x)) \tag{9.22}$$

Figure 9.14 shows how the multi objective optimization selects one of the feasible short-term plans. The blue short-term plans in Figure 9.14a are the short-term plans that do not violate any hard constraints. The green short-term plan in Figure 9.14b is the short-term plan with the highest score, and is therefore selected.



(a) 200 generated short-term plans from STP-R where the short-term plans that violate the hard constraints are red, and the short-term plans that fulfills the hard constraints are blue.

(b) The result after applying the soft constraints. The red short-term plans violates one of the hard constraints. The blue short-term plans fulfill the hard constraints. The green short-term plan is the short-term plan that achieves the highest score from the multi objective optimization.

Figure 9.14.: The process of applying the multi objective optimization with the soft constraints on a set of short-term plans that fulfills the hard constraints.

9.8. Planning in different driving situations

The weights that are used in the multi objective optimization determines the behaviour of the autonomous vehicle. The weight for the soft risk constraint w_1 determines how fast the autonomous vehicle should approach other objects. By adjusting this w_1, it is possible to determine how aggressive the autonomous vehicle should act. If w_1 is high, then the autonomous vehicle is penalized for proposing risky short-term plans. This will lead to a vehicle which waits at intersections until other moving objects have crossed. The weight for the soft comfort criteria w_2 influences the shape of the possible short-term plans. By instance, if w_2 is high, then the autonomous vehicle will not propose to overtake other vehicles. This is due to the fact that an overtaking creates some jerk.

The target speed in the environment is an upper limit, and do not consider the shape of the road. This is a realistic setting, and human drivers reduces the speed in sharp curves to make the curve comfortable. The weight for the soft constraint w_3 determines how close to the speed limit that is preferable. An autonomous vehicle should drive close to the speed limit on straight roads, but reduce the speed in sharp curves, and hence deviate from the target speed. By selecting a high w_3, the autonomous vehicle might not be able to obtain this behavior. The weight for the high-level plan criteria w_4 has some of the same effect as w_1. If w_4 is high then the autonomous vehicle will not overtake other vehicles since these short-term plans deviates from the high level plan.

This reasoning shows the necessity to change the weights based on different situations. We have observed four different general situations that require different behaviours from the ego vehicle. The situations are categorized by the presence of moving objects, and if the ego vehicle is in an intersection or not. This categorization results in four different groups of situations that require different weights. The driving situation is easliy determined by the current navigational command and the existence of other moving objects in the dynamic Circogram. The driving situations are illustrated together with the corresponding weights in Figure 9.15. The weights are selected based on experiments and visually analyzing the resulting behavior.

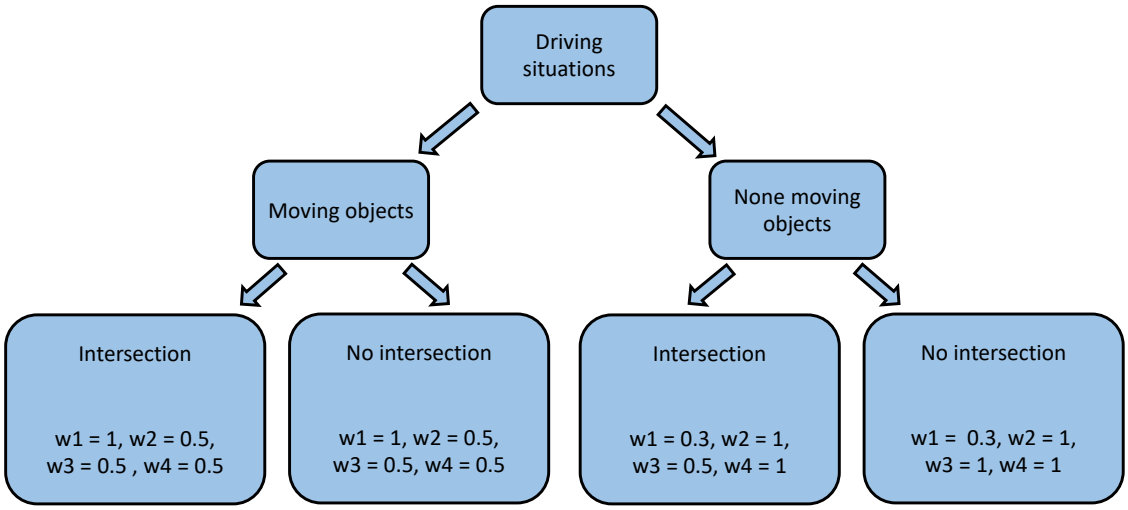


Figure 9.15.: Different optimization weights for the different soft constraints for the different driving situations.

Different weights depending on the defined situations will affect the short-term plan selection described in section 9.7, the short-term planner with imitation learning described in subsection 9.5.2, and the short-term planner with reinforcement learning described in subsection 9.5.3. The short-term plan selection process will only have one minor change. Instead of keeping the weights constant, they will change depending on the current driving situation. By only changing the short-term plan selection process, one assume that the local plan parameters are equal in all the situations. This might not be the case, and the local plan parameters could be generated by different policies.

Inspired by Ohn-Bar et al. 2020, we create four different policies which depend on the driving situation. The imitation learning setup explained in subsection 9.5.2 is used to train the neural net-

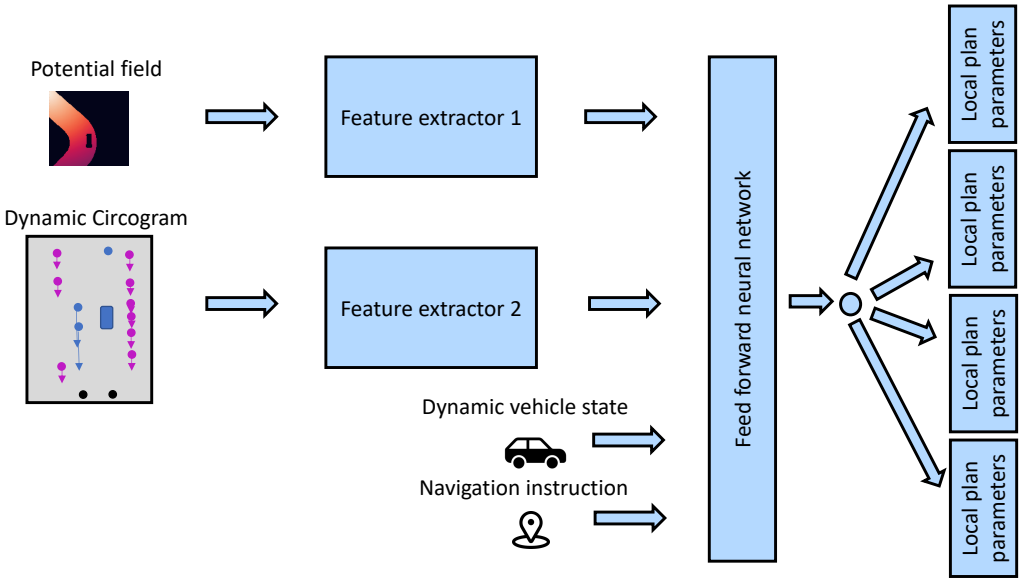


Figure 9.16.: An extension of the neural network policy to allow for different policies in different driving situations. This neural network architecture has four different heads that are conditioned on the driving situation.

work. The only difference is that the neural network has four heads that condition on the situation. This means that all the policies use the same feature extractor, and only the heads differ. This is illustrated in Figure 9.16. The four heads require target values that comes from different driving behaviours. Targets that reflect the four different driving behaviours are created by the random generation of parameters with fine tuning, using the weights in Figure 9.15 together with ρ_4 (closeness to the segmented high-level plan). The four heads are trained on each of the resulting four datasets, one at the time. The correct head is selected base on the current driving situation when the autonomous vehicle is operating. This result in local plan parameters that reflect the current driving situation. The short-term plan selection is also performed with the weights in Figure 9.15. The autonomous vehicle obtains more diverse behaviours by conditioning on the driving situation in the component that creates the local plan parameters.

Another approach to create local plan parameters from policies that depend on the driving situation is to adjust the STP-RL explained in subsection 9.5.3. This can be done be creating four different reward functions based on Figure 9.15. The reward functions only differ in the weights that are used for each of the soft short-term plan selection criteria. This will result in four different policies that differ in risk, comfort, and how strictly they follow the high-level plan. This has not been implemented, but similar a approach is done by Yang, Sun, and Narasimhan (2019).

10. Plan Execution and Control

10.1. Control to follow short-term plan

Control theory can handle the control of dynamical systems in engineered processes and machines. The goal is to develop a model that adjusts the system inputs to drive the system to a desired state, while minimizing any delay, overshoot, or steady-state error. This is highly relevant for an autonomous vehicle which for example should adjust the throttle to keep the speed limit and contain a desired distance to surrounding objects. In the autonomous vehicle literature, it has been used one controller for the throttle and one for the steering commands. A longitudinal controller is used to control the throttle commands, and a lateral controller is used to control the steering commands.

10.1.1. PID controller

A PID controller is a controller that uses feedback to try to minimize the error between the actual state and the desired reference state. It corrects the error by applying a proportional, a integrated, and a derivative term. The proportional term handles the current error, the integral term is used to handle the past errors, and the derivative term is used to handle the effect of future errors. This is expressed in Equation 10.1.

$$u(t) = K_p e(t) + K_i \int_0^\infty e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{10.1}$$

where the error $e(t) = |r(t) - y(t)|$ is the difference between the reference state and the actual state. K_p, K_i and K_d, all non-negative, denote the coefficients for the proportional, integral, and derivative terms respectively. To discretize Equation (10.1) is needed to select a way of discretizing the integral $i[k]$ between the time indices $k-1$ and k, and a way of discretizing the derivatie $d[k]$ between $k-1$ and k. The time between $k-1$ and k is T_s seconds. The time interval between each command will be the same as the time between each frame in the simulation. The integral is descretized with the trapezoidal rule, and the derivative is discretized with a finite differnce approximation. The discretization of the integration term in the PID-controller is given by,

$$i[k] = i[k-1] + T_s \frac{e[k] + e[k-1]}{2}. \tag{10.2}$$

The discretization of the derivative term in the PID-controller is given by,

$$d[k] = \frac{e[k] - e[k-1]}{T_s} \tag{10.3}$$

The complete discretization of Equation (10.1) is given by,

$$u[k] = K_p e[k] + K_i i[k] + K_d d[k] \tag{10.4}$$

10.1.2. Stanley controller

The *DARPA Grand Challenge (DGC)* is an collection of challenges where teams have the opportunity to test autonomous vehicles in a competitive situations [Behringer et al. 2004]. In addition to intelligent behaviour, the participating vehicles must also exhibit ruggedness and endurance in order to survive the fast ride over rough terrain. The Stanford team won the 2005 DGC with a car called Stanley. The lateral controller they developed for this car has later been called the Stanley controller [Hoffmann et al. 2007]. The controller is a lateral geometric controller that has shown better performance than the simple pure pursuit controller. It uses the center of the front axels as a reference point, and uses both the heading error and the distance from the closest point on the trajectory to steer. The goal of the controller is to correct the heading error, the position error and to stay within the maximum steering angels.

$$\delta(t) = \psi(t) + tan^{-1}(\frac{ke(t)}{k_s + v_f(t)}) \tag{10.5}$$

where $\delta(t)$ is the steering angel at time t, $\psi(t)$ is the heading error, $e(t)$ is the positional error, $v_f(t)$ is the speed, and k and k_s are two different gains. The discrete version of Equation (10.6) at the discrete time k is given by,

$$\delta[k] = \psi[k] + tan^{-1}(\frac{Ke[k]}{K_s + v_f[k])}) \tag{10.6}$$

10.1.3. Tuning of gains for the controllers

The longitudinal and lateral controllers tracks the best short-term plan. The gains and the softening constant used in the experiments are presented in Table 10.1. These have been tuned manually to obtain the desired behaviour.

Parameter name	**Value**
Proportional gain, K_p	0.07
Integral gain, K_i	0.3
Derivative gain, K_d	0.09
Steering gain, K	0.2
Softening constant, K_s	0.01

Table 10.1.: Controller gain constants that are used in the longitudinal and lateral controllers.

Figure 10.1 shows that the controller is able to track an arbitrary short-term plan created by the planning component. The resulting path from the composite controller seems to cut the curves.

(a) Snapshots of the ego vehicle which tracks the short-term plan.

(b) Comparison of the short-term plan and the executed path. The green line is the short-term plan, and the black line is the executed path.

Figure 10.1.: A vehicle that tracks a short-term plan by using lateral and longitudinal controllers. The green line shows an arbitrary short-term plan, and the snapshots of the vehicle shows that the vehicle is able to follow the short-term plan.

10.2. Execute control action sequence

The short-term plan is based on a sequence of discrete actions called a control action sequence, and can be directly used to control the vehicle. If the motion model that creates the short-term plans corresponds with the simulation environment, then the vehicle will follow the short-term plan by executing the discrete actions. As showed in section 8.5, the complete physical motion model creates plans that are similar to the measured movement from a sequence of discrete actions. Hence, executing the control action sequence that underlies the short-term plan will make the vehicle follow the short-term plan. Figure 10.2 shows that the ego vehicle is able to track an arbitrary short-term plan. The resulting path from executing the control action sequence is a bit shorter than the short-term plan, which means that the average speed is lower than in the short-term plan. The performance in this example is similar to the performance in section 8.5. The similarity between the short-term plan and the executed path depends on how well the physical motion model correspond to the vehicle in the simulation.

(a) Snapshots of the ego vehicle which tracks the short-term plan.

(b) Comparison of the short-term plan and the executed path. The green line is the short-term plan, and the black line is the executed path.

Figure 10.2.: A vehicle that tracks a short-term plan by executing actions from the control action sequence. The green line shows an arbitrary short-term plan, and the snapshots of the vehicle shows that the vehicle is able to follow the short-term plan.

11. Experimental Results

The results from the autonomous vehicle are shown in this chapter. section 11.1 starts by showing the parameters and the configurations that are used for the experiments. Then the route that is designed to test the SafeRide system is presented, followed by the results in this route. This section presents images of the path of the ego vehicle.

11.1. Experimental setup of the SafeRide system

Several different configurations for each component can be used throughout the design of the SafeRide system. Initial experiments have been conducted during the design of the SafeRide system. The initial experiments are used to decide some of the configurations that are used in the testing of the system. Appendix I shows all the initial experiments that have been conducted, and a small comment is written for every experiment. All the configurations and parameters used to create the experimental results are presented in this section. The CARLA simulation is run in synchronous mode at 30 frames per second.

11.1.1. Sensor configurations

A semantic camera is used to create the recolored BEV sensor. The semantic camera is placed 7 m above and 4 meters in front of the ego vehicle, and has a pitch angle of -90 degrees. This means that the camera is facing directly down 4 meters in front of the ego vehicle. Further, the images consist of $512x512$ pixels and the field of view is set to 125 degrees.

11.1.2. Environment representation configuration

The dynamic Circogram uses 100 simulated rays that are equally distributed around the ego vehicle. The potential field is created from the same recolored BEV that is used to create the dynamic Circogram. The potential field does not need as many pixels as the $512x512$ recolored BEV contains. Therefore, the potential field is downsampled with max-pooling with a stride equal to the filter size. Experiments have shown that a $128x128$ size is sufficient. This size of the potential field is therefore used in the experimental results.

11.1.3. Selection of motion model

chapter 8 presents three different motion models that take the dynamic vehicle state as input. In the experiments, we will use the CTRV model explained in section 8.1. This choice is made because of the simple implementation, and because Schubert, Richter, and Wanielik (2008) shows that the complexity of the motion model is not significant for vehicle tracking. The motion model calculates the dynamic vehicle states at time intervals of $\frac{1}{60}$ seconds.

Driving situation	Number of data samples
Intersection no moving objects	5000
Intersection with moving objects	4000
No intersection no moving objects	10000
No intersection with moving objects	3000

Table 11.1.: The amount of data that is used to train the short-term planner based on imitation learning for each driving situation. The driving situations are the same as in section 9.8.

11.1.4. Setup of the planning component

The general parameters in the planning component are presented in Table 11.2. The STP-IL is the short-term planner that is used in these experiments. The complete architecture of the autoencoder network is listed in Table E.1, and the complete architecture of the policy network is listed in Table G.1. These tables include the activation functions, the initialization of the weights, the output shape of each layer, the kernel shape, the stride, the padding, and the number of parameters for each layer. The neural networks have been trained on 21000 training samples. The distribution of the training samples between the different driving situations is shown in Table 11.1. The weights presented in Figure 9.15 are used to adjust the behaviour depending on the driving situation.

Parameter name	Value
Short-term plan frequency, ω	0.5 [s]
Length of short-term plan, T	3 [s]
Number of short-term plans, N	50
Number of local plan parameters	4
Standard deviation throttle local plan parameter, $\sigma_{\mathbf{r}}$	0.1
Standard deviation steering local plan parameter, $\sigma_{\mathbf{z}}$	0.1

Table 11.2.: Parameters used in the experiments for the planning component. The parameters that do not have a unit are unitless.

11.1.5. Execution of the best short-term plans

In chapter 10, two different ways of executing the best short-term plan are proposed. In the experiments in this section, we have chosen to execute the control action sequence. This is the preferred way of tracking the short-term plan since it removes the need of introducing a controller that requires tuning.

11.2. Experimental testing route

A route that requires different maneuvers is developed to test the SafeRide system described in section 11.1. The route is between two locations in Town02 which has not been used for training. Town02 is similar to Town01 but smaller. The testing route is illustrated in Figure 11.1. The figure shows the start location marked with a green circle, different driving situations marked with blue circles, and the goal location marked with a red circle. All the circles have a corresponding overview image of the situation that the ego vehicle needs to handle. The high-level plan described in subsection 9.2.2 is also plotted to show where the desired direction. Figure H.1 shows a bigger figure of the complete testing route where the driving situations are stacked together.

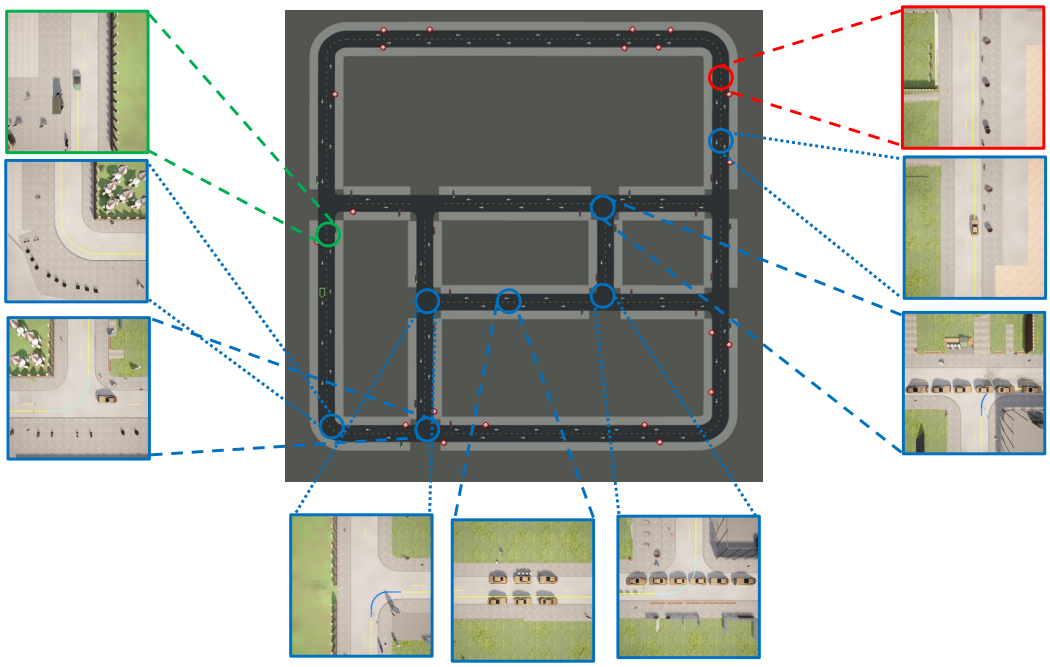


Figure 11.1.: The testing route that is designed to test the trained SafeRide system. The test route consists of seven different situations that required different maneuvers of the autonomous vehicle. The situations in the boxes around the map will be arrived in a counter clockwise manner, starting from the box at the top to the left, and ending at the box at the top to the right.

The testing route starts at the green circle in Figure 11.1. The autonomous vehicle needs to follow the lane until a sharp left turn while the navigational command still is to follow the lane. Then the next driving situation is that the vehicle needs to take a left turn at the next intersection. This needs to be done while another static vehicle is placed at the other side of the intersection. This requires that the autonomous vehicle notices that the other vehicle is not moving, and it is safe to cross the other lane. The following driving situation is a right turn at the next intersection. Thereafter, the ego vehicle is approaching parked cars on each side of the road. None of the vehicles is moving, and the gap between the cars makes it possible to drive between them. This illustrates a narrow passage that the ego vehicle needs to drive through. At the next intersection, there will be another moving vehicle in the intersection when the ego vehicle needs to make a decision. The other moving vehicle needs to start driving at a particular time to reach the intersection as an obstacle for the ego vehicle. The selection of the appropriate starting time is done by iterative adjustments. The next intersection includes a similar driving situation, but this time the other moving vehicle is approaching the intersection from another direction. The starting time of this vehicle is found by the same process as the other dynamic obstacle vehicle. The last situation is a static vehicle in the middle of the lane. Then the route ends at the red circle in Figure 11.1. The results from these seven driving situations will be presented in the next section.

11.3. Results from different driving situations in the test route

In this section, the results from all the different driving situations in the test route illustrated in Figure 11.1 are presented. An image containing snapshots of the vehicles is used to show the performance of the agent in a specific driving situation.

11.3.1. Left turn when navigation command is to follow lane

The first driving situation in the test route is a 90 degrees left turn. This turn is not part of an intersection, which means that the navigational command is to follow the lane. This driving situation will reveal if the autonomous vehicle detects the turn from the hitpoints in the Circogram, and the features from the potential field. The results from the left turn when the navigational command is to follow the lane are shown in Figure 11.2.

Figure 11.2.: The first driving situation in the testing route. This shows the result of a left turn when navigation command is to follow lane. Seven snapshots of the ego vehicle are stitched together to this figure.

The result shows that the ego vehicle is driving close to the curb before the curve starts. The ego vehicle starts to turn around 8 meters before the beginning of the turn, which is a few meters after the hitpoints in the Circogram indicate that there is a turn. The ego vehicle touches the lane markings in the middle of the turn before it is approaching the curb again. At the end of the turn, the ego vehicle is again close to the curb. The ego vehicle drives through the curve with approximately the same velocity, and the sharp turn at the end of the curve creates some additional jerk. This shows that the ego vehicle is able to follow a lane that includes curves.

11.3.2. Left at intersection with static car

The second driving situation is the first time that the ego vehicle needs to make a decision about where to drive. As seen in Figure 11.1, the navigational command is to take a left turn at this intersection. In addition, there is placed another vehicle in the opposite lane on the other side of the intersection. This vehicle is static, and does not move during the test route. The static vehicle is placed close enough to the intersection to be detected in the Circogram by the ego vehicle when it is close to the intersection. The result from this intersection is shown in Figure 11.3.

Figure 11.3.: The second driving situation in the testing route. This shows the result of left at intersection with static car. Seven snapshots of the ego vehicle are stitched together to this figure.

Once again the result shows that the ego vehicle is driving close to the curb when it is following the lane, and the road is straight. The ego vehicle starts to turn at the correct time and keeps a safe distance from the other static vehicle. Also in this turn, the ego vehicle touches the other lane in the middle of the curve. This shows that the agent is able to select the correct road in an intersection and that it is able to do this while other moving objects are detected.

11.3.3. Right at intersection without other cars

The third driving situation is a right turn in an intersection. The right turns are sharper than the left turns due to the right-hand traffic in the CARLA towns. No other moving objects are present in this driving situation, and it only tests the performance of the ego vehicle at right turns in intersections. The result from this intersection is shown in Figure 11.4.

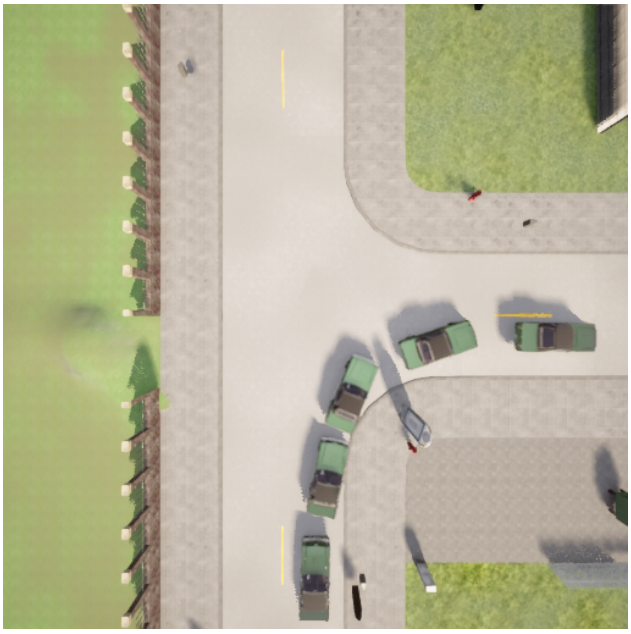


Figure 11.4.: The result of right at intersection without other vehicles. This driving situation contains the sharpest curve the ego vehicle need to drive. Five snapshots of the ego vehicle are stitched together to this figure.

Figure 11.4 shows that the ego vehicle is very close to the curb at the beginning of the curve. It does not hit the curb, but it is closer to the curb than the defined safety margin that the ego vehicle should keep to other objects. The ego vehicle needs to adjust the course to not hit the curb. It drives straight for a short time before it again turns right. At the end of the curve the ego vehicle is close to the lane marking.

11.3.4. Narrow passage through parked cars at both sides of the road

This driving situation checks how the ego vehicle operates when the free space of the road gets narrower. In this driving situation, there are placed three static cars on each side of the road. The width of the road is reduced by four meters, and the narrow part of the road is in total 15 meters. The result from this driving situation is shown in Figure 11.5.

Figure 11.5.: The result of driving through a narrow passage consisting of parked cars at both sides of the road. Six snapshots of the ego vehicle are stitched together to this figure.

The first snapshot of the ego vehicle reveals that the ego vehicle is still close to the lane marking after the previous driving situation. The orientation of the ego vehicle shows that it is facing the curb, and would probably approach the curb if no obstacles are present. The ego vehicle turns to the left to avoid hitting the first of the parked cars on the right side of the road. Thereafter, the ego vehicle drives as close as possible to the parked cars to the right without causing a collision. After the last parked car, the ego vehicle turns to the right to drive in the intended lane.

11.3.5. Left at intersection with dynamic car

This driving situation is the first with another dynamic car. The other dynamic car starts when the ego vehicle is at a specified distance from the interaction, and drives with a constant throttle. This is defined to ensure that the other vehicle will become an obstacle for the ego vehicle in the interaction. The ego vehicle needs to cross the driving lane to the other dynamic vehicle to follow the navigational command. The result of this driving situation is shown in Figure 11.6. This result consists of two figures since the paths are crossing.

(a) The first part of the driving situation of a left turn at intersection with another dynamic car. Five snapshots of the vehicles are stitched together to this figure. The ego vehicle is only present at three positions in the figure since it is not moving in the last two snapshots.

(b) The second part of the driving situation of a left turn at intersection with another dynamic car. Two snapshots of the ego vehicle are stitched together to this figure. The other dynamic vehicle is not present since the ego vehicle does not start to drive before it is outside of this area.

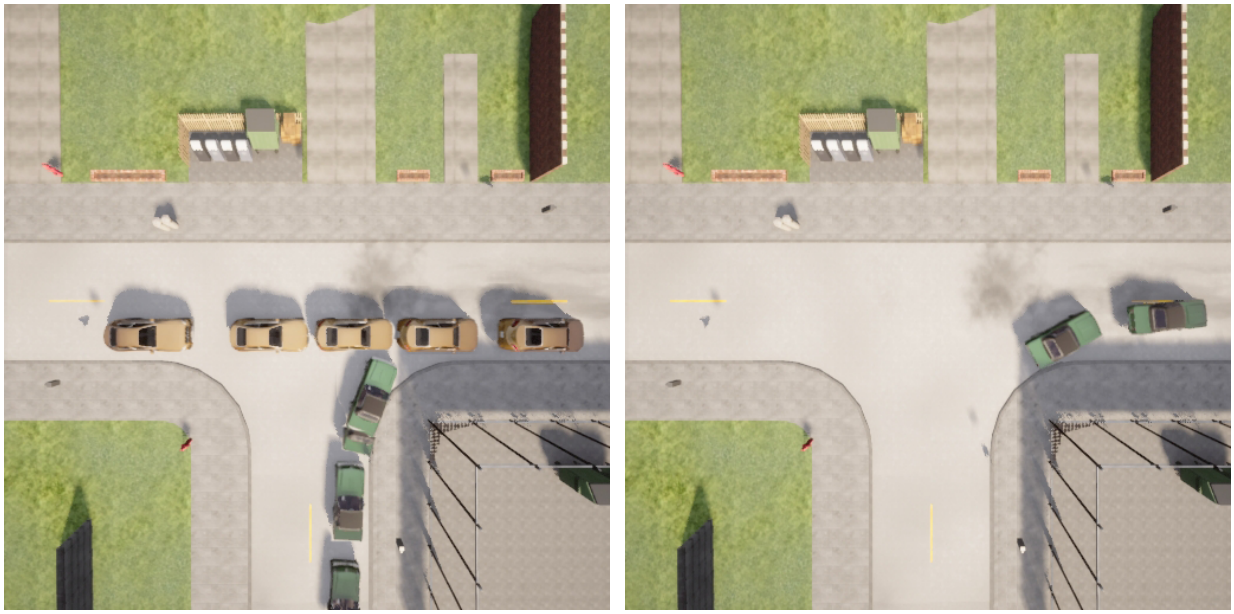
Figure 11.6.: The result of right at intersection with dynamic car creating an obstacle. This driving situation is illustrated with two subfigures since the paths from the the vehicles overlap.

Figure 11.6a consists four snapshots of the area. In the first snapshot, only the other vehicle is visible. The three next snapshots contain both of the vehicles. The distance between the two first ego vehicle positions is larger than the distance between the position of the other vehicle. This is because the ego vehicle has a higher velocity than the other vehicle. The third and fourth snapshots of the ego vehicle are overlapping, meaning that the ego vehicle has slowed down its speed. In the last snapshot, the ego vehicle is very close to the other vehicle, but they have not collided. The ego vehicle can slow down to avoid hitting another moving vehicle when it needs to cross the lane.

Figure 11.6b is an extension of the driving situation in Figure 11.6a. The two first snapshots of the ego vehicle are almost in the same position. The ego vehicle starts to drive slowly after the other vehicle has passed. After the two first snapshots, half of the other vehicle is outside of the figure. The ego vehicle increases the speed after the two first snapshots and completes the left turn at the intersection.

11.3.6. Right at intersection with dynamic car

This driving situation differs from the previous since the desired lane to the ego vehicle is the same that the other dynamic car is using. The ego vehicle either needs to drive in front of the other dynamic car, or slow down and drive behind the other dynamic car. Once again, the other dynamic car starts when the ego vehicle is at a specified distance from the interaction, and drives with a constant throttle. Figure 11.7 shows how the ego vehicle drives in this driving situation.



(a) The first part of the driving situation of a right turn at intersection with another dynamic car. Five snapshots of the vehicles are stitched together to this figure. The ego vehicle is only present at three positions in the figure since it is not moving in the last two snapshots.

(b) The second part of the driving situation of a right turn at intersection with another dynamic car. Two snapshots of the ego vehicle are stitched together to this figure. The other dynamic vehicle is not present since the ego vehicle does not start to drive before it is outside of this area.

Figure 11.7.: The result of right at intersection with dynamic car creating an obstacle. This driving situation is illustrated with two subfigures since the paths from the the vehicles overlap.

Figure 11.7a shows the first part of the driving situation. This figure contains five snapshots of the area, and both of the vehicles are in the area during all the snapshots. It looks like the ego vehicle only appears in three of the snapshots, but in the three last snapshots, the ego vehicle does not change its position significantly. The ego vehicle stops just in front of the other dynamic vehicle and avoids a collision. Figure 11.7b is an extension of the driving situation in Figure 11.7a. This figure only contains two snapshots and shows that the ego vehicle starts to drive when the other car is outside the area.

11.3.7. Parked car in the middle of the lane

The final driving situation in the test route is a parked car that is located in the middle of the appropriate lane for the ego vehicle. The car is parked on a straight road, and no other vehicles are influencing the driving situation. The result from this driving situation is shown in Figure 11.8.

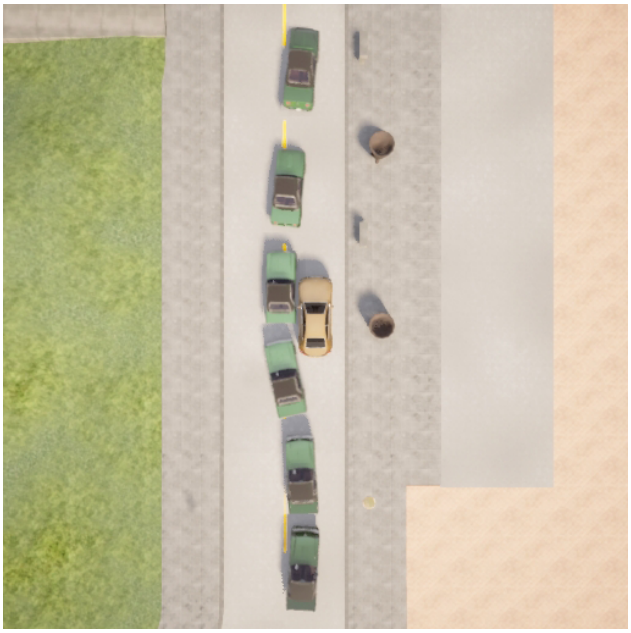


Figure 11.8.: The result of overtaking a parked car in the middle of the lane. Six snapshots of the ego vehicle are stitched together to this figure.

This situation is located close to an intersection where the ego vehicle has turned left. The first snapshot of the ego vehicle shows that it is oriented towards the curb. This is probably since the ego vehicle has not had time after the previous intersection to get close to the curb. The next snapshot of the ego vehicle shows that the orientation of the ego vehicle has changed, and the ego vehicle has decided to overtake the parked car. This happens when the gap to the parked car is five meters. Although the ego vehicle decides to overtake the parked car early, it still drives very close to the parked car when it overtakes. After the ego vehicle has overtaken the parked car, it returns to its lane.

12. Discussion of the Design Process and the Results

The discussion consists of two parts. In the first part, the initial experiments and the design process are discussed. These experiments have influenced the design, and the selection of parameters used in the final experiment in the test route. The second part of the discussion addresses all the different driving situations in the test route illustrated in Figure 11.1.

12.1. Discussion of the initial experiments during the design process

This section discusses initial experiments completed during the design process. The SafeRide system has been developed with an iterative process. The initial experiments are described in Appendix I. This spreadsheet contains information about the setup of the experiment and a short comment on important observations. This overview of the initial experiments shows that we started with a minimum viable design, and added complexity to overcome the unwanted behaviour of the agent. The goal of the SafeRide system is to create a small set of feasible short-term plans based on a sparse intermediate environment representation, where the best short-term plan can be selected.

12.1.1. Discussion of minimal viable agent

In the first initial experiments, I only used the static part of the Circogram, generated random local plan parameters from a uniform distribution, removed short-term plans that caused a collision, and selected the best of the remaining short-term plans based on the closeness to the segmented high-level plan and the speed limit. The result of this design is an agent that is available to drive without causing a collision in a static environment. However, the agent creates many short-term plans causing a collision and proved the inefficiency of the one-step STP-R. The small set of short-term plans that did not cause a collision disabled the functionality of the short-term plan selection. The fine-tuning of the best local plan parameters from the one-step STP-R was introduced to increase the set of short-term plans which complies with the hard constraints. This results in more short-term plans that the agent can use in the short-term optimization.

12.1.2. Discussion of the number of local plan parameters

The same setup was used to investigate the influence of the number of parameters to use in the DAG. We used a planning horizon of 3 seconds, and tested different ways of creating piecewise linear functions for the steering and throttle input within this time horizon. We tested 2, 3, and 4 parameters for each input. Additionally, we tested the possibility of having time parameters linked to the input parameters. These different ways of parameterizing the DAG decide the complexity of the short-term plans. By adding more parameters, the agent can output more complex short-term plans. Our observation is that more than 2 parameters for each command input create unstable short-term plans, and unnecessary complexity. For a short time interval, it is sufficient to adjust the slope of the control action sequence a few times. However, it might be beneficial to use more

parameters in more complex situations. For instance, one could create a finite state machine that decides how many parameters the DAG should use depending on the driving situation.

12.1.3. Discussion of comfort and speed limits

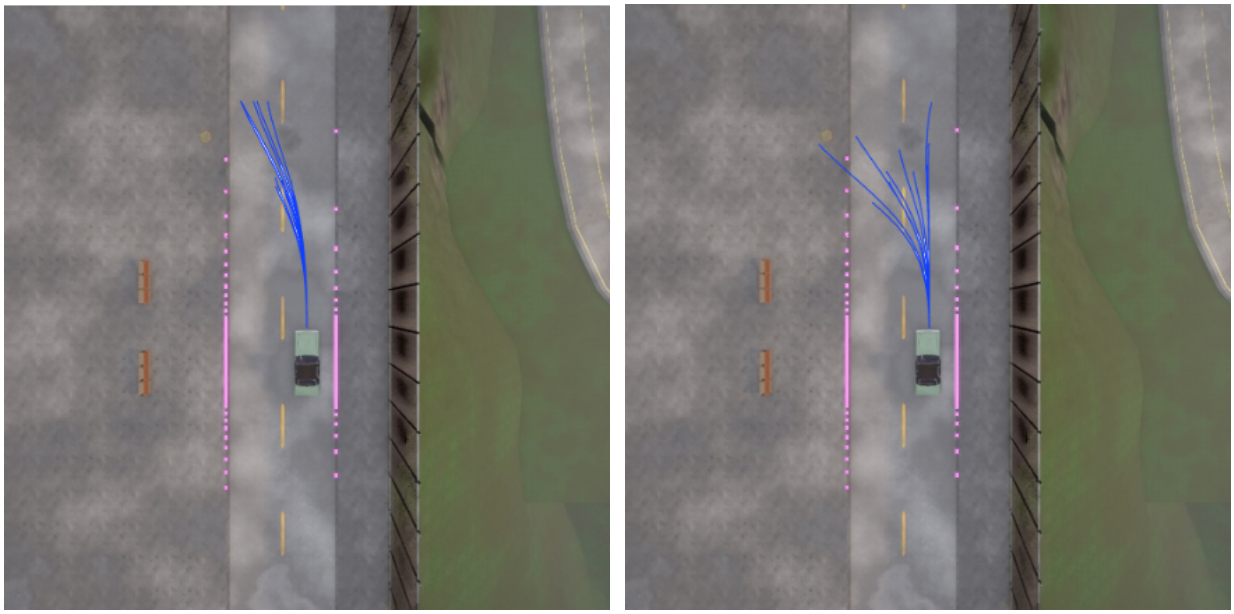
The autonomous vehicle with only collision avoidance as hard constraints, and closeness to the speed limit and the segmented high-level plan had two noticeable issues. The vehicle did not comply with the speed limit, and it had curvy behaviour on straight roads. To overcome these issues, we added a hard constraint on the speed limit and a soft constraint on the jerk. This resulted in an autonomous vehicle that never drives faster than the speed limit, and soft constraint on the jerk stabilized the short-term plans and hence increased the comfort. A hard constraint on the speed limit is very effective, and this makes it impossible for the ego vehicle to drive faster than the speed limit. This is the desired behaviour most of the time, but in some unusual situations, there might be beneficial to drive a bit faster than the speed limit. This might be necessary to avoid some dangerous situations. Therefore, the final decision was to adjust the hard constraint to be 5% above the speed limit. Another way to solve this issue might be to have the hard constraint on the speed limit, and remove this hard constraint in dangerous situations where it might increase safety.

12.1.4. Discussion of safety margin to obstacles

With the hard and soft constraints discussed in the previous paragraphs, the autonomous vehicle was able to drive simple routes without any obstacles. Therefore, the complexity of the simulation was increased by adding static and dynamic vehicles. The ego vehicle was able to overtake other static obstacles, but it drives very closely to the obstacles. This is due to the soft constraint on the closeness to the segmented high-level plan which is in the middle of the desired lane. It is risky to drive close to objects that can move. For example, a passenger might open a door on another vehicle and cause a collision, or the vehicle might suddenly start to drive. As an action to handle this, a small safety margin was added to the collision check. This makes the vehicle increase the distance to all objects that can cause a collision. Nevertheless, this is not exactly the desired behaviour since it is not risky to closely pass objects that cannot move. This might prohibit the autonomous vehicle to drive on narrow roads with oncoming traffic. The dynamic Circogram contains information about the semantic class to the hitpoints, and this could be used to only create a safety margin around objects that might move. A way to overcome this is to replace the safety margin in the hard constraint with a soft constraint on the distance to objects that might move. This will not restrict the accessibility of the autonomous vehicle and simultaneously preserve safety. Another measure to increase the safety is soft risk criterion based on TTC. It was included to avoid colliding with other dynamic vehicles, and to be able to differentiate between driving towards a wall and behind another vehicle with the same velocity as the ego vehicle. The soft risk criterion also had an unintended benefit, in fact, it improved the ego vehicle's ability to perform right turns. Without the soft risk criteria, the ego vehicle made too wide right turns, and often ended up in the wrong lane after the turn. In these wide turns, the ego vehicle is driving towards to curb on the other side of the road which results in a low TTC. Therefore, the soft risk criterion made the ego vehicle quickly change the direction and avoided wide right curves.

12.1.5. Discussion of the short-term plan selection

As explained in subsection 9.7.3, the weighting of the soft constraints will heavily determine the behaviour of the ego vehicle. Extensive testing of different combinations of weights has been conducted, and the best weights were found by visually inspecting the resulting behaviour of the ego



(a) A standard deviation of $\sigma_z = \sigma_r = 0.05$ when sampling a set of 10 local plan parameters that the DAG generates short-term plans of with the physical motion model.

(b) A standard deviation of $\sigma_z = \sigma_r = 0.2$ when sampling a set of 10 local plan parameters that the DAG generates short-term plans of with the physical motion model.

Figure 12.1.: Illustration of different confidence levels in the local plan parameters. A small standard deviation is interpreted as high confidence in the output from the learning-based short term planners, and a large standard deviation is treated as the opposite.

vehicle and by defining the desired characteristic in different driving situations. For instance, in the current system the weights will decide if the ego vehicle will use the opposite lane to overtake a parked vehicle.

The size of the standard deviations σ_z and σ_r for the local plan parameters determine how important the short-term plan selection is, and hence also the selection of the weights. As explained in subsection 9.5.2 and subsection 9.5.3, the local plan parameters that the learning-based short-term planners produce are assumed to be mean values for normal distributions. By having a large corresponding standard deviation in the normal distribution, the sampling of short-term plans will result in a more diverse set of short-term plans. This means that the short-term plan selection is more influential on which short-term plan the agent selects. On the other hand, a small corresponding standard deviation will result in many similar plans and the short-term plan selection process will not be as important. Therefore, the size of the standard deviation can be interpreted as the confidence in the local plan parameters from the short-term planners. Figure 12.1 illustrates this interpretation. If the confidence in the correctness of the local plan parameters is high, the standard deviation could be low. However, for safety reasons, the standard deviations should be above a certain threshold to make sure that there is always a feasible short-term plan to follow. There are some uncertainties in learning-based approaches, and there is reason to treat the output as random variables.

12.1.6. Discussion of the learning-based short-term planners

In section 9.5, two different learning-based short-term planners are proposed. One uses imitation learning from prerecorded driving, and one optimizes a reward function using reinforcement learning.

The idea was to train the policy network in Figure 9.7 with imitation learning first, and then improve the robustness and generalization through reinforcement learning. I was not able to improve the policy with reinforcement learning, and this will probably improve the agent. During the collection of prerecorded data, different scenarios were recorded through the traffic manager in CARLA. The gathering of training data for the STP-IL is time-consuming due to the creation of the target values (the true local plan parameters). The target values were gathered from an STP-R with a DAG that generated a large set of feasible short-term plans. The other approaches described in Equation (9.7) are independent of the rest of the SafeRide system, and might result in better target values. The STP-IL will probably improve if the set of prerecorded driving is enlarged. The hyperparameters in the policy network are not thoroughly tuned. The generalization might be improved with a different network architecture. However, with the possibilities for improvements in the generation of local plan parameters, the SafeRide system is able to complete a testing route with different curves, static obstacles, and dynamic obstacles.

12.2. Discussion of the results from the test route

In this section, each of the driving situations in the testing route is discussed in chronological order. Both the desired characteristics and the shortcomings of the agent are discussed.

12.2.1. Left turn when navigation command is to follow lane

The result from the left turn when the navigational command is to follow the lane is shown in subsection 11.3.1. The information that indicates that the ego vehicle needs to turn in this situation is the extracted features from the potential field and the hit points in the Circogram. The custom recolored BEV sensor is set up such that it detects objects 30 meters in front of the ego vehicle. The result from this driving situation shows that it is sufficient to detect the curve 30 meters prior to the curve to successfully complete it with the current speed limit. This might not be the case for higher speed limits.

This shows that the ego vehicle curvature of the path increases throughout the curve, before the agent suddenly starts to drive straight when the curve is ending. A path that has a curvature that starts at 0 and increases linearly with the path length is defined as a clothoid. These paths have been shown to reduce the lateral jerk during a curve, and are heavily used in railway engineering. Clothoids have also been used in optimizing the racing line during the corner entry portion of a turn [Hossain, Eager, and Walker 2020]. This is a good characteristic of the path in the curve, and it might be a consequence of soft constraint on the lateral jerk. However, the sudden change in curvature at the end of the curve might cause some jerk and be uncomfortable. Paths that look like clothoids also create some drawbacks. The clothoids shape of the path makes the ego vehicle cross the opposite lane in the middle of the curve. This might be a dangerous path if the ego vehicle meets another vehicle in the curve. Nevertheless, it is not possible to discuss the behaviour of the ego vehicle in such a situation without seeing how it behaves. The ego vehicle might drive on clothoids when no vehicle in the opposite lane is detected, and completely stay in its lane when it meets other vehicles in the curve.

12.2.2. Left in intersection with static car

The result from the left turn in the intersection with a static car is explained in subsection 11.3.2. The snapshots of the ego vehicle show that the ego vehicle drives as if the other vehicle is not existing. In this situation, this is the desired behaviour since the other car is static. This shows that

the agent has generalized to some degree since it is able to ignore the static car. The ego vehicle drives close to the curb before and after the intersection. This might be done to create a curve with a bigger radius, which increases the comfort.

However, the velocity seems to be fairly constant during the whole curve. The comfort and the decision making could be improved if the ego vehicle has slowed down before the intersection. By slowing down before the interaction, the lateral jerk through the curve will decrease. It might also be easier to stop if the other vehicle suddenly starts to drive.

12.2.3. Right in intersection without other cars

The result from the right intersection without other cars is described in subsection 11.3.3. This result shows that the ego vehicle also can handle sharp turns in the CARLA town. In this intersection, the ego vehicle ends up close to the other lane. This reduces the sharpness of the curve but might be risky if other vehicles are driving in the opposite lane.

12.2.4. Narrow passage through parked cars at both sides of the road

The result from the narrow passage through parked cars on both sides of the road is described in subsection 11.3.4. This driving situation illustrates some of the shortcomings of the agent. The agent can drive through the narrow passage without colliding, but it drives close to the parked cars on the right side of the lane. The size of the passage is too small for two cars with the size of the ego vehicle to pass. Therefore, the ego vehicle should treat it as one lane and could drive on top of the lane markings to increase safety. It is no risk connected with driving close to non-movable objects, but the cars on the right side of the road could start to drive. This justifies the previous proposal of adding the distance to movable objects as a soft constraint.

Although, this behaviour is a result of the creation of the target values for the STP-IL. The targets are selected based on the closeness to the segmented high-level plan which is in the middle of the lane the ego vehicle is supposed to drive. This reflects an agent with STP-IL who prefer driving close to the middle of the lane when it is possible. Another way of overcoming this shortcoming is to create the targets for the STP-IL from an autopilot instead of STP-R. The autopilot will not drive that close to the parked cars on the right side of the road. The collection of training data for the STP-IL could also be created from manual driving. This would probably give the best training set to learn from, but this is a very time-consuming process.

The final way to improve this behaviour is to use STP-RL instead of STP-IL. The reward function needs to reflect this desired behaviour, and the agent needs to be exposed to these situations several times.

12.2.5. Left in intersection with dynamic car

The result from the left turn in the intersection with a dynamic car is described in subsection 11.3.5. The ego vehicle stops very close to the other dynamic car. This is a consistent characteristic of the SafeRide system, and should be adjusted. Such driving might cause insecurity by the other drivers, which might respond by trying to keep a bigger distance. The ego vehicle starts to drive when the other dynamic vehicle is out of the path that the ego vehicle needs to drive to follow the navigational command. This shows the agent is able to detect when the obstacle has disappeared, and the ego vehicle continues driving at this moment. For the efficiency of the traffic, this is an important aspect.

The STP-IL has access to the relative velocity between the outer surface of the vehicle and the hit points from the simulated rays in the Cricogram. This means that the STP-IL have information that makes it possible to create local plan parameters that take the movement of the dynamic car into account. Nevertheless, this driving situation required the other dynamic car to start within a small specified time interval for the ego vehicle to avoid a collision. If the other dynamic car starts a second later, then the ego vehicle attempts to cross the intersection before the other dynamic car, which results in the other dynamic car colliding with the rear end of the ego vehicle. The current version of SafeRide is not robust in driving situations with other moving objects. This can be solved by simulating the movement of the hitpoints with velocity when performing the hard constraint collision check, or by including more diverse data for the STP-IL. The first suggestion is to improve the short-term plan selection process, and the second proposal is to improve the generation of local plan parameters.

The first proposal to improve the robustness of driving situations with moving objects is to extend the collision checking. All the needed information to perform this extension is already in the dynamic Circogram. The velocity of each hit point with class label *movable object*, could be assumed to have a constant velocity, and used to calculate paths that could be used in the collision check. This means that it is required to perform a collision check at specified intervals during the length of the short-term plan. Each position of the ego vehicle along with the proposed plan, needs to be checked for collision with the calculated position of the movable hit points at the same discrete time instance. This extended collision check increases the computational time of the short-term plan selection.

The second proposal to improve the robustness of driving situations with moving objects is to add more diverse training data for the STP-IL. As explained in section 11.1, the STP-IL is trained on prerecorded driving from the STP-R with information about the high-level plan from CARLA. The STP-R only stops when all the generated short-term plans lead to a collision. Hence, it is not possible to use this way of creating training data to increase the robustness of the STP-IL with other dynamic objects. One possible solution is to use the STP-R to collect training data in simple settings without other dynamic objects, and use an autopilot or drive the vehicle manually in difficult situations with other dynamic obstacles.

12.2.6. Right in intersection with dynamic car

The result from the right turn in the intersection with a dynamic car is described in subsection 11.3.6. The discussion around the robustness with other dynamic objects also applies to this driving situation. Therefore, this part of the discussion is not repeated here to avoid redundancy. However, it is possible to observe one advantage with the current collision check constraint. The ego vehicle does not start to drive before it can create a short-term plan that does not lead to a collision along the complete planned path. This result in a safety margin of the planning length. It is a rule of thumb that one should keep a safety margin of two seconds in some countries[1]. The ego vehicle is forced to keep a safety margin of the planning length by checking the complete short-term plan for collision in the current static Circogram. This analysis shows that one should both check the complete short-term plan for collision in the static Circogram, and use the velocity of the hitpoints in the Circogram to calculate the temporal development of the hit points and check these for collision.

[1] *NYS DMV - Driver's Manual - Chapter 8: Defensive Driving* refer to this as the two-second-rule: `https://dmv.ny.gov/about-dmv/chapter-8-defensive-driving`

12.2.7. Parked car in the middle of the lane

The result from overtaking the parked car in the middle of the lane is described in subsection 11.3.7. This result is similar to the result in subsection 11.3.4. The only differences are that in this driving situation there is one other car, and the car is placed in the middle of the preferred driving lane to the ego vehicle. This shows that the ego vehicle can deviate further away from the generated high-level plan from CARLA. The same shortcomings that were discovered in subsection 11.3.4 also apply here, and will not be discussed again to avoid redundancy.

13. Conclusion and Further Work

This chapter concludes the thesis and proposes some further work to improve the SafeRide system.

13.1. Conclusion

In this thesis, we present a novel structured approach to autonomous vehicles in simulated environments that can be tested and certified. The system is called SafeRide and is developed in the simulation environment CARLA. The structured approach consists of an intermediate environment representation that is created from a semantic bird´s eye view. The intermediate environment representation is sufficient to drive the testing route. This intermediate environment representation is used together with the dynamic vehicle state and a navigational command to create a small set of local plan parameters. The dynamic Circogram and the potential field contain sufficient information for driving in the test environment. The local plan parameters are used to create control action sequences, which are transformed into short-term plans by a physical vehicle motion model. This shows that it is possible to create short-term plans fulfilling nonholonomic constraints based on control action sequences. To the best of my knowledge, the use of sequences of control actions to create short-term plans has not been researched before. This thesis also shows that it is possible to design a physical vehicle motion model in CARLA which reflects how the simulated vehicles move. The short-term plans are tested for hard constraints and optimized based on soft constraints. The weights in the multi objective optimization are highly influencing the behaviour of the autonomous car. The best control action sequence that results in the best short-term plan can be executed or the short-term plan can be executed by a controller that compensates for the deviations between the short-term plan and the observed motion.

The final autonomous vehicle can drive smoothly through random routes that contain obstacles. Since the short-term plans are based on a physical vehicle motion model, it is feasible for the autonomous vehicle to correctly follow the proposed plans. This thesis shows that it is possible to create short-term plans that precisely describe where the vehicle is going to drive. Some other approaches to short-term planning are based on geometric curves, which may not be feasible for a car to execute due to nonholonomic constraints. The final autonomous vehicle can avoid obstacles when driving and follow navigational instructions similar to what GPS-based systems offer. The results show that the autonomous vehicle drives on paths similar to clothoids in the curves. Such curves are comfortable for the passengers.

This structured approach is selected because it makes it possible to test and certify each component. The regulators demand this feature of the autonomous driving system[1]. The SafeRide system includes the possibility to adjust the confidence in the learning-based short-term planner by adjusting the standard deviations when sampling local plan parameters. It is easy in the SafeRide

[1] ISO 22737: Intelligent transport systems — Low-speed automated driving (LSAD) systems for predefined routes — Performance requirements, system requirements and performance test procedures: `https://www.iso.org/standard/73767.html`

system to analyse each of the components to verify the performance and find possibilities for improvements.

13.2. Further Work

The natural extension of this thesis is to make the system more robust in dynamic environments. Better robustness in dynamic environments can either be done by generating better local plan parameters, or to extend the path selection to handle dynamic environments. To create better local plan parameters, it is necessary to improve the learning-based short-term planner. The short-term planner with imitation learning can be improved by collecting more and better training data. Better training data can be collected by using another approach for calculating the target values. This can be done by using an autopilot, or by manual driving. The better training data set will be more time-consuming to collect than the current training data set. As mentioned in this thesis, imitation learning is criticized for lacking robustness and training data bias. This is still an issue if the current training data set is improved. These limitations can be handled by using reinforcement learning to train the short-term planner. This was the intended final step of training the short-term planner, but this training phase was not conducted due to the time constraints of the thesis.

The second way of adapting the current system to a dynamic environment is to adapt the short-term plan selection. This can be done by simulating the movements of the hit points in the Circogram during the short-term plan length. To perform this extension, it is necessary to make some assumptions about the movements of the hit points. A simple assumption that could be used as a starting point is to assume that the hit points move with a constant velocity.

Another observed characteristic that can be improved is how the ego vehicle behaves around other movable objects. In the current system, the ego vehicle behaves the same way around static objects and movable objects that have zero velocity. Movable objects with zero velocity might suddenly start to move, and it is therefore riskier to be close to such objects. This can be addressed by adding a new soft constraint that penalizes the closeness of the movable objects. This will only influence the short-term plan selection process, and it might be more beneficial to make an adjustment that influences the creation of the local plan parameters. One way to influence the creation of the local plan parameters is to add additional information in the environment state that can be used to learn the intended new relation. This can be done by adding repulsive forces around movable objects in the potential field. It can be combined with the velocity to add stronger repulsive forces in the direction the other vehicles drive.

Each of the modules in the SafeRide system required an extensive design process, and it was not time to fine-tune the neural network architecture. It might be able to improve the performance of the short-term planner by changing the neural network architecture for the policy. The neural network can be changed in different ways. The current neural network architecture is relativly small due to the small training data set. It was kept small to avoid overfitting to the small training data set. A bigger training data set will allow the designer to use a larger neural network. The number of hidden layers and the number of neurons in each hidden layer can be increased.

Bibliography

[1] Aaslund, Bjørn André. "Simulated Autonomous Driving in Nordic Climate — Preproject". In: (2021).

[2] Badino, Hernán, Franke, Uwe, and Pfeiffer, David. "The Stixel World - A Compact Medium Level Representation of the 3D-World". In: *Pattern Recognition*. Vol. 5748. Springer Berlin Heidelberg, Sept. 2009, pp. 51–60. DOI: 10.1007/978-3-642-03798-6_6.

[3] Behringer, R. et al. "The DARPA grand challenge - development of an autonomous vehicle". In: *IEEE Intelligent Vehicles Symposium, 2004*. 2004, pp. 226–231. DOI: 10.1109/IVS.2004.1336386.

[4] Bojarski, Mariusz et al. "End to End Learning for Self-Driving Cars". In: (Apr. 2016). URL: https://www.researchgate.net/publication/301648615_End_to_End_Learning_for_Self-Driving_Cars.

[5] Bøhn, Eivind et al. "Optimization of the Model Predictive Control Update Interval Using Reinforcement Learning". In: *IFAC-PapersOnLine* 54.14 (2021). 3rd IFAC Conference on Modelling, Identification and Control of Nonlinear Systems MICNON 2021, pp. 257–262. DOI: https://doi.org/10.1016/j.ifacol.2021.10.362.

[6] Chen, Dian, Koltun, Vladlen, and Krähenbühl, Philipp. "Learning to drive from a world on rails". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 15570–15579. DOI: 10.1109/ICCV48922.2021.01530.

[7] Chen, Dian et al. "Learning by Cheating". In: *Proceedings of the Conference on Robot Learning*. Ed. by Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, 2020, pp. 66–75. URL: https://proceedings.mlr.press/v100/chen20a.html.

[8] Chitta, Kashyap, Prakash, Aditya, and Geiger, Andreas. "NEAT: Neural Attention Fields for End-to-End Autonomous Driving". In: *International Conference on Computer Vision (ICCV)*. 2021. DOI: 10.48550/ARXIV.2109.04456.

[9] Codevilla, Felipe et al. "Exploring the Limitations of Behavior Cloning for Autonomous Driving". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 9328–9337. DOI: 10.1109/ICCV.2019.00942.

[10] Dosovitskiy, Alexey et al. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1–16. URL: https://proceedings.mlr.press/v78/dosovitskiy17a.html.

[11] Falanga, Davide et al. "PAMPC: Perception-Aware Model Predictive Control for Quadrotors". In: *IEEE/RSJ International Conference on Intelligent c Robots and Systems (IROS), Madrid*. Apr. 2018, pp. 1–8. DOI: 10.1109/IROS.2018.8593739.

[12] Hayati, Hasti et al. "Jerk within the Context of Science and Engineering—A Systematic Review". In: *Vibration* 3.4 (2020), pp. 371–409. DOI: 10.3390/vibration3040025.

[13] Hoffmann, Gabriel et al. "Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing". In: *2007 American Control Conference*. Aug. 2007, pp. 2296 –2301. DOI: 10.1109/ACC.2007.4282788.

[14] Hossain, Md Imam, Eager, David, and Walker, Paul. "Greyhound racing ideal trajectory path generation for straight to bend based on jerk rate minimization". In: *Scientific Reports* 10 (Apr. 2020), p. 7088. DOI: 10.1038/s41598-020-63678-1.

[15] Hwang, Ching-Lai and Masud, Abu Syed Md. "Methods for Multiple Objective Decision Making". In: *Multiple Objective Decision Making — Methods and Applications: A State-of-the-Art Survey*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1979, pp. 21–283. DOI: 10.1007/978-3-642-45511-7_3.

[16] Kendall, Alex et al. "Learning to Drive in a Day". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 8248–8254. DOI: 10.1109/ICRA.2019.8793742.

[17] Kiran, Bangalore et al. "Deep Reinforcement Learning for Autonomous Driving: A Survey". In: *IEEE Transactions on Intelligent Transportation Systems* PP (Feb. 2021), pp. 1–18. DOI: 10.1109/TITS.2021.3054625.

[18] Klose, Patrick and Mester, Rudolf. "Simulated Autonomous Ariving in a Realistic Driving Environment using Deep Reinforcement Learning and a Deterministic Finite State Machine". In: *Proceedings of the 2nd International Conference on Applications of Intelligent Systems* (Jan. 2019), pp. 1–6. DOI: 10.1145/3309772.3309802.

[19] Kong, Jason et al. "Kinematic and dynamic vehicle models for autonomous driving control design". In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. June 2015, pp. 1094–1099. DOI: 10.1109/IVS.2015.7225830.

[20] Koren, Yoram and Borenstein, Johann. "Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2. May 1991, pp. 1398 –1404. DOI: 10.1109/ROBOT.1991.131810.

[21] Liang, Xiaodan et al. "CIRL: Controllable Imitative Reinforcement Learning for Vision-Based Self-driving". In: *Computer Vision – ECCV 2018*. Ed. by Vittorio Ferrari et al. Cham: Springer International Publishing, 2018, pp. 604–620. DOI: https://doi.org/10.1007/978-3-030-01234-2_36.

[22] Liu, Shuying and Deng, Weihong. "Very deep convolutional neural network based image classification using small training sample size". In: *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*. 2015, pp. 730–734. DOI: 10.1109/ACPR.2015.7486599.

[23] Miettinen, Kaisa. "A Priori Methods". In: *Nonlinear Multiobjective Optimization*. Boston, MA: Springer US, 1998, pp. 115–129. DOI: 10.1007/978-1-4615-5563-6_5.

[24] Ohn-Bar, Eshed et al. "Learning Situational Driving". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020, pp. 11293–11302. DOI: 10.1109/CVPR42600.2020.01131.

[25] Paden, Brian et al. "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles". In: *IEEE Transactions on Intelligent Vehicles* 1 (Apr. 2016). DOI: 10.1109/TIV.2016.2578706.

[26] Pepy, R., Lambert, A., and Mounier, H. "Path Planning using a Dynamic Vehicle Model". In: *2006 2nd International Conference on Information Communication Technologies*. Vol. 1. 2006, pp. 781–786. DOI: 10.1109/ICTTA.2006.1684472.

[27] Philion, Jonah and Fidler, Sanja. "Lift, Splat, Shoot: Encoding Images From Arbitrary Camera Rigs by Implicitly Unprojecting to 3D". In: *Proceedings of the European Conference on Computer Vision*. 2020. DOI: DOI:10.1007/978-3-030-58568-6_12.

[28] Prakash, Aditya, Chitta, Kashyap, and Geiger, Andreas. "Multi-Modal Fusion Transformer for End-to-End Autonomous Driving". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 7073–7083. DOI: 10.1109/CVPR46437.2021.00700.

[29] Rajamani, Rajesh. *Vehicle dynamics and control*. 2nd ed. Springer New York, 2012. DOI: https://doi.org/10.1007/978-1-4614-1433-9.

[30] Rawlings, J and Mayne, D.Q. *Model Predictive Control: Theory and Design*. 2nd ed. Nob Hill Publishing, LLC, Jan. 2009.

[31] Roddick, Thomas and Cipolla, Roberto. "Predicting Semantic Map Representations from Images using Pyramid Occupancy Networks". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 11135–11144. DOI: 10.1109/cvpr42600.2020.01115.

[32] Roth, Scott D. "Ray casting for modeling solids". In: *Computer Graphics and Image Processing* 18.2 (1982), pp. 109–144. DOI: https://doi.org/10.1016/0146-664X(82)90169-1.

[33] Schubert, Robin, Richter, Eric, and Wanielik, Gerd. "Comparison and evaluation of advanced motion models for vehicle tracking". In: *2008 11th International Conference on Information Fusion*. Jan. 2008, pp. 1–6. DOI: 10.1109/ICIF.2008.4632283.

[34] Silver, David et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (Jan. 2016), pp. 484–489. DOI: 10.1038/nature16961.

[35] Song, Yunlong and Scaramuzza, Davide. "Learning High-Level Policies for Model Predictive Control". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Nov. 2020. DOI: 10.1109/IROS45743.2020.9340823.

[36] Song, Yunlong and Scaramuzza, Davide. "Policy Search for Model Predictive Control With Application to Agile Drone Flight". In: *IEEE Transactions on Robotics* (2022), pp. 1–17. DOI: 10.1109/TRO.2022.3141602.

[37] Toromanoff, Marin, Wirbel, Emilie, and Moutarde, Fabien. "End-to-End Model-Free Reinforcement Learning for Urban Driving using Implicit Affordances". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 7151–7160. DOI: 10.1109/cvpr42600.2020.00718.

[38] Yang, Runzhe, Sun, Xingyuan, and Narasimhan, Karthik. "A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/4a46fbfca3f1465a27b210f4bdfe6ab3-Paper.pdf.

A. Definitions Used in the Thesis

The thesis contains many definitions and created names. All these definitions are presented in this chapter together with a short explanation. There is reference to the section in the thesis where they are used the first time. The tables are divided into CARLA definitions, sensor definitions, environment representation definitions, physical motion model definitions, and planning definitions. They are presented sequentially in own tables.

Name	**Explanation**
Spawing position	A location in the CARLA simulation where the ego vehicle can be teleported without causing a collision. Used in section 4.4.

Table A.1.: Important names relevant to CARLA together with a short explanation.

Name	**Explanation**
Recolored BEV	A semantic bird´s eye view of the ego vehicle where some classes have been merged. Used in section 6.6.

Table A.2.: Important defined names relevant to the sensors together with a short explanation.

Name	**Explanation**
Simulated ray	A straight line in an image which ends when it hits objects. Simulates how an laser ray works. Used in section 7.1.
Hit point	A point where the simulated ray collides with a non-drivable object. Used in section 7.1.
Circogram	An intermediate environment representation that contains distances to the boundary of the free space. Used in section 7.1.
Dynamic Circogram	Extension of the Circogram where relative velocity and the semantic class of the boundary of the free space are included. Used in section 7.1.
Potential field	An intermediaste environment representation which encode a temporary goal pose and the free space as a matrix with positive values. Used in section 7.2.

Table A.3.: Important defined names relevant to the intermediate environment representation component together with a short explanation.

Name	Explanation
Dynamic vehicle state	The state vector variable used in the motion models. It consists of . Used in chapter 8.

Table A.4.: Important defined names relevant to the physical motion model component together with a short explanation.

Name	Explanation
High-level plan	A plan with information about how to reach the goal without considering the dynamics of the ego-vehicle and surrounding objects. Used in section 9.3.
Navigational command	A discrete variable which describes which road to drive to get to the goal location. Used in subsection 9.2.1.
Segmented high-level plan	A segment of the sequence of equally distributed points from the start location to the goal location. Each point in the segment also contain a navigation command. Used in subsection 9.2.2.
Attraction point	A high-level plan that only consists of one point. The attraction point is the point in the sequence of equally distributed points from the current location of the ego vehicle to the goal location where the navigational command change. It is a point that consists of a position and the navigational command that should be used until this location. Used in subsection 9.2.2.
Control action sequence	A sequence segment of control actions (steering z and throttle r) for a short time period into the future. Used in section 9.4
Local plan parameters	Some parameters describing the steering input z and the throttle input r at some time in the future. Used in subsection 9.4.1.
Driving action generator (DAG)	A component that transforms local plan parameters to control action sequences. Used in subsection 9.4.1.
Short-term plan	A plan describing the motion of the ego vehicle a short time into the future. The motion model takes control action sequences as inputs and outputs short-term plans/trajectories. Used in section 9.4.
Short-term planner	A component that creates local plan parameters from the current dynamic Circogram, the vehicle state, and the current high-level plan. Used in section 9.5.
Environment state	An environment state in this chapter correspond to the input to the short-term planner. The state consists of the current dynamic Circogram, the vehicle state, and the current high-level plan. Not confused with the dynamic vehicle state presented in chapter 8. Used in section 9.5.
Plan execution module	Executes the short-term plans/trajectories. Used in section 9.4 but explained in chapter 10.

Table A.5.: Important defined names relevant to the planning component together with a short explanation.

B. Affine Transformation between Coordinate Frames

The creation of the Circogram is happening in the pixel coordinate frame, defined with $(0, 0)$ in the upper left corner, x being the row number, and y being the column number. Thus, it is necessary to be able to transform the pixel coordinates to the vehicle coordinate frame. The BEV camera is always pointing vertically down to the ground. It is possible to use an affine transformation to transform coordinates between these coordinate systems, if we assume that the ground is flat. This will be true for the roads in the CARLA towns that we use. An augmented matrix and vector is used since it makes it possible to represent both the translation and the linear map using a single matrix multiplication.

$$\begin{bmatrix} y \\ 1 \end{bmatrix} = \left[\begin{array}{c|c} A & b \\ 0 \dots 0 & 1 \end{array}\right] \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{B.1}$$

16 points distributed on the road around the vehicle was selected, and the corresponding pixel location was found. Least squares method is used to to find the best parameters.

Figure B.1.: Points that are used to create the affine transformation between the coordinate frames.

C. Difficulties of Creating the Circogram from a BEV in CARLA

The Circogram is created by sending simulated rays from the center of gravity of the ego vehicle. The hitpoints are found by traversing the simulated ray and detecting the first pixel outside the ego vehicle that is not a drivable area. This procedure works well as long as the ego vehicle has a shape without any major objects protruding from the surface. This can for example be big side mirrors or wheels that are turned maximum to one side. Such protruding objects might cause the simulated ray to detect a hitpoint on its own surface. This needs to handled to avoid such events. The Mustang that is used in this thesis has small side mirrors that do not influence the simulated rays, but the front wheels collide with the simulated rays when the steering wheel is turn maximum to the left or the right. This is handled by adjusting the starting pixels for the simulated rays that are affected.

D. Discontinuity of Angles

All angles in this thesis are in radians or transformed to radians when obtained from CARLA. Angles in radians will have a discontinuity of 2π. Since we consider the angles to be $\theta \in [-\pi, \pi]$, we will have a discontinuity between $-\pi$ and π. The removal of the phase discontinuity is completed by the following steps. Given an input angle θ close to the discontinuity. Subtract π from the angle to get away from the discontinuity. Then compute the sine and cosine representation of the angle, and then go back into the angle representation by using $atan2$. Finally add π to get into the same value range, if needed. $\hat{\theta}$ is the angle without the discontinuity.

$$\hat{\theta} = atan2(\sin(\theta - \pi), \sin(\theta - \pi)) + \pi \tag{D.1}$$

Figure D.1 shows the orientation with discontinuity, and the result after using Equation D.1

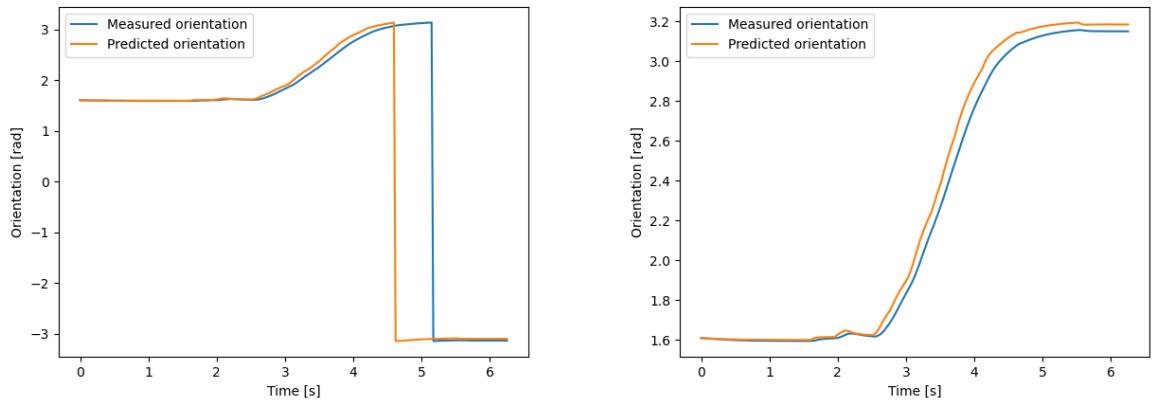


Figure D.1.: The orientation θ before the removal of the discontinuity, and the result $\hat{\theta}$ after the removal of the discontinuity.

E. Complete Autoencoder Architecture

Layer	Act.	Init.	Output	Kernel	Stride	Padding	Param
0_pf_input	-	-	(1x128x128)	-	-	-	0
1_pf_conv2d	Relu	He	(8x42x42)	(5, 5)	(3, 3)	(1, 1)	0
2_pf_conv2d	Relu	He	(16x14x14)	(5, 5)	(3, 3)	(1, 1)	0
3_pf_conv2d	Relu	He	(32x4x4)	(5, 5)	(3, 3)	(1, 1)	0
4_pf_flatten	Relu	He	(512)	-	-	-	0
5_pf_linear	Relu	He	(128)	-	-	-	
6_pf_linear	Relu	He	(512)	-	-	-	
7_pf_unflatten	-	-	(32x4x4)	-	-	-	0
8_pf_convtranspose2d	Relu	He	(16x14x14)	(5, 5)	(3, 3)	-	0
9_pf_convtranspose2d	Relu	He	(8x42x42)	(5, 5)	(3, 3)	(1, 1)	0
10_pf_convtranspose2d	Relu	He	(1x128x128)	(5, 5)	(3, 3)	-	0
Total params: 8,765							
Trainable params: 8,765							

Table E.1.: Model summary for the autoencoder network used to extract features from the potential field. The table lists the activation function (Act.), the weight initialization (Init.), the output shape (Output), the kernel shape (Kernel), the stride, the padding, and the number of parameters each layer contains.

F. Recreated images from the trained autoencoder

(a) Original potential field.

(b) Recreated potential field.

(c) Original potential field.

(d) Recreated potential field.

(e) Original potential field.

(f) Recreated potential field.

Figure F.1.: Original potential fields and the recreated potential fields from the trained autoencoder. These images show that the autoencoder is able to preserve the temporary goal pose and the area of the free space.

G. Complete Policy Architecture

Layer	Act.	Init.	Output	Kernel	Stride	Padding	Param
0_pf_input	-	-	(1x128x128)	-	-	-	0
1_pf_conv2d	Relu	He	(8x42x42)	(5, 5)	(3, 3)	(1, 1)	0
2_pf_conv2d	Relu	He	(16x14x14)	(5, 5)	(3, 3)	(1, 1)	0
3_pf_conv2d	Relu	He	(32x4x4)	(5, 5)	(3, 3)	(1, 1)	0
4_pf_flatten	Relu	He	(512)	-	-	-	0
5_pf_linear	Relu	He	(128)	-	-	-	
0_dc_input	-	-	(4x100)	-	-	-	0
1_dc_conv1d	Relu	He	(8x100)	(5)	(1)	(1)	0
2_dc_conv1d	Relu	He	(8x100)	(5)	(1)	(1)	0
3_dc_maxpool1d	-	-	(8x25)	(4)	(4)	-	0
4_dc_conv1d	Relu	He	(16x25)	(5)	(1)	(1)	0
5_dc_conv1d	Relu	He	(16x25)	(5)	(1)	(1)	0
6_dc_maxpool1d	-	-	(16x6)	(4)	(4)	-	0
7_dc_flatten	-	-	(96)	-	-	-	0
0_ffn_concatenate	-	-	(232)	-	-	-	0
1_ffn_linear	Relu	He	(64)	-	-	-	0
2_ffn_linear	Tanh	Xavier	(4)	-	-	-	0
Total params: 8,765							
Trainable params: 8,765							

Table G.1.: Model summary for policy network used for mapping environment states to local plan parameters. The table lists the activation function (Act.), the weight initialization (Init.), the output shape (Output), the kernel shape (Kernel), the stride, the padding, and the number of parameters each layer contains.

H. The testing route

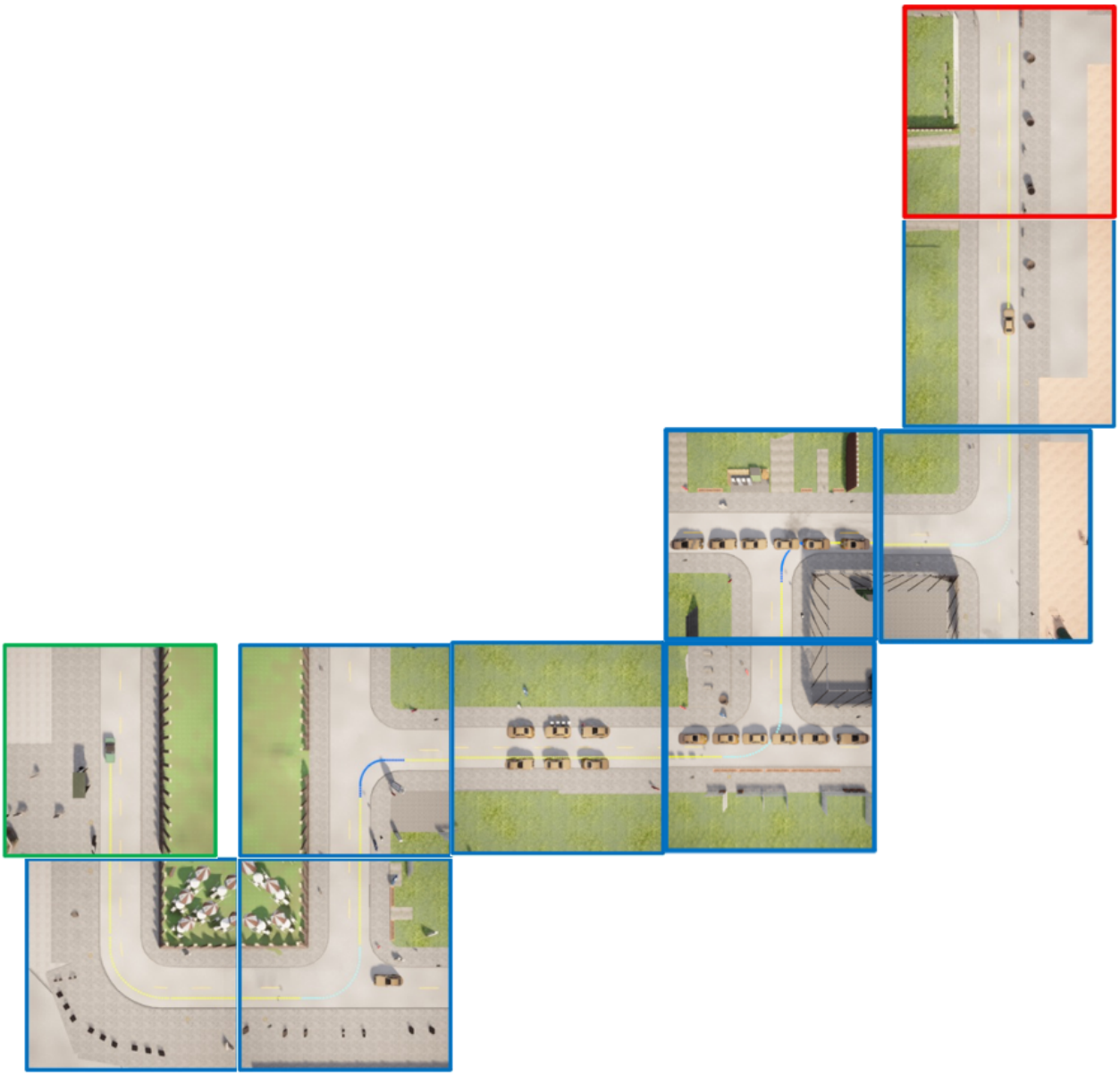


Figure H.1.: The complete testing route where all the driving situations are connected. Parts of the route where the ego vehicle only need to follow the lane are removed. This makes the high-level plan provided by CARLA a continuous line from the starting position to the end position.

I. Initial experiments during the design process of SafeRide

General parameters						Parameter selection			
Movie name	Path length	FPS	Number of parameters	Number of generated trajectories	Setting	Uniform random	Uniform random and finetuned	Imitation learning	RL
26042022_random_1.mov	3	30	5 (z1, z1, r1, r2, t1)	50	No other vehicels	x			
26042022_random_finetuned_1.mov	3	30	3 for z and r	50	No other vehicels		x		
26042022_random_finetuned_2.mov	3	30	3 for z and r	50	No other vehicels		x		
26042022_random_finetuned_3.mov	3	30	2 for z and r	50	No other vehicels		x		
26042022_random_finetuned_4.mov	3	30	4 for z and r	50	No other vehicels		x		
26042022_random_finetuned_5.mov	3	30	2 for z and r	100	No other vehicels		x		
26042022_random_finetuned_6.mov	3	30	2 for z and r	100	One static car		x		
26042022_random_finetuned_7.mov	3	30	2 for z and r	100	No other vehicels		x		
26042022_random_finetuned_8.mov	3	30	2 for z and r	100	Dynamic environment		x		
28042022_random_finetuned_1.mov	3	30	2 for z and r	200	Dynamic environment (too many cars)		x		
28042022_random_finetuned_2.mov	3	30	2 for z and r	200	Dynamic environment (too many cars)		x		

Path selection hard constraints		Path selection soft constraints								
Collision	Speed limit	Target velocity (w1)	Distance from high level plan (w2)	Total jerk (w3)	TTC (w4)	Distance to atractor (w5)	Weights	Comment
x		x	x					Few trajectories that are used for path selection
x		x	x					Curvy driving on straights.
x		x	x					Curvy driving on straights. Drives very fast for some periods.
x		x	x					Stable driving on straights. Cuts two corners.
x		x	x					Very curvy driving. Cuts one corner.
x	x	x	x	x				Very stable. Sometimes to big turns.
x	x	x	x	x				Passes the car very closely.
x	x	x	x	x			w1 = 1.4	Still to wide turns. Do not cur corners.
x	x	x	x	x			w1 = 1.4	Handles static and dynamic cars well.
x	x	x	x	x			w2 = 1.4	Very hard environment. w2 is so high that it stops behind one other car. Cuts one corner.
x	x	x	x	x			w2 = 1.1	Collides with one car that comes from left. Shows the need for TTC.

	General parameters					Parameter selection			
Movie name	Path length	FPS	Number of parameters	Number of generated trajectories	Setting	Uniform random	Uniform random and finetuned	Imitation learning	RL
28042022_random_finetuned_3.mov	3	30	2 for z and r	200	Dynamic environment		x		
28042022_random_finetuned_4.mov	3	30	2 for z and r	200	Dynamic environment		x		
30042022_random_finetuned_1.mov	3	30	2 for z and r	50	Dynamic environment		x		
01052022_random_finetuned_1.mov	3	30	2 for z and r	200	Dynamic environment		x		
01052022_random_finetuned_2.mov	3	30	2 for z and r	200	Dynamic environment		x		
02052022_random_finetuned_1.mov	3	30	2 for z and r	100	Dynamic environment		x		
02052022_imitation_learning_1.mov	3	30	2 for z and r	100	No other vehicels			x	

Path selection hard constraints		Path selection soft constraints									
Collision	Speed limit	Target velocity (w1)	Distance from high level plan (w2)	Total jerk (w3)	TTC (w4)	Distance to atractor (w5)	Weights		Comment		
x	x	x	x	x			w2 = 1.2		Town03. Stops the simulation before a tunnel. Works well with highways.		
x	x	x	x	x			w2=1.1, w3=0.6		Looks good. Good behaviour in the outer lane in curves.		
x	x	x	x	x	x		w2=1.1, w3=0.6		First with TTC. Inner lane in curves look better. Closer to the right in inner lane curve. Slows down when meeting other cars. Need to reduce the TTC weight.		
x	x	x	x	x	x		w2=1.1, w3=0.6, w3=0.6		Tries to stop behind a car, but crashes gently.		
x	x	x	x	x	x		w1 = 1, w2 = 1, w3 = 1, w4 = 1		The driving behaviour meets the expectations.		
x	x	x	x	x	x		w1 = 1, w2 = 1, w3 = 1, w4 = 2		Shows that the ego vehicle stops when two other cars are blocking the road.		
x	x	x	x	x	x		w1 = 1, w2 = 1, w3 = 1, w4 = 1		Need more data for the learning. Only have 9000 samples		
									Forgot to one hot encode the high level plan, which might causing the issue in the turns. Need more data for the		

General parameters						Parameter selection			
Movie name	Path length	FPS	Number of parameters	Number of generated trajectories	Setting	Uniform random	Uniform random and finetuned	Imitation learning	RL
05052022_imitation_learning_1.mov	3	30	2 for z and r	100	No other vehicels			x	
05052022_imitation_learning_2.mov	3	30	2 for z and r	100	No other vehicels			x	
06052022_imitation_learning_1.mov	3	30	2 for z and r	100	No other vehicels			x	
06052022_imitation_learning_2.mov	3	30	2 for z and r	100	No other vehicels			x	
06052022_imitation_learning_3.mov	3	30	2 for z and r	100	No other vehicels			x	
10052022_imitation_learning_1.mov	3	30	2 for z and r	100	No other vehicels			x	

Path selection hard constraints		Path selection soft constraints									
Collision	Speed limit	Target velocity (w1)	Distance from high level plan (w2)	Total jerk (w3)	TTC (w4)	Distance to atractor (w5)	Weights		Comment		
x	x	x	x	x	x		w1 = 1.1, w2 = 0.8, w3 = 0.6, w4 = 0.6		Looks better now. Currently have 15000 samples. Added more data from curves.		
x	x	x	x	x	x		w1 = 1.1, w2 = 0.8, w3 = 0.6, w4 = 0.6		Looks better now. Currently have 15000 samples. Added more data from curves.		
x	x	x	x	x	x		w1 = 1.1, w2 = 0.8, w3 = 0.6, w4 = 0.6		More data, 18000. One intersection, and one turn on follow lane.		
x	x	x	x	x	x		w1 = 1.1, w2 = 0.8, w3 = 0.6, w4 = 0.7		Other intersections, still good.		
x	x	x		x	x		w1 = 1, w3 = 1, w4 = 1		Removed soft constraint on high level plan segment. The result is that the car does not keep its lane. It also fails in the intersection.		
x	x	x		x	x	x	w1 = 1, w3 = 1, w4 = 1, w5=1		Only use the distance to the attraction point as a soft constraint. This makes it follow the lane. Able to do turn when it follows the lane, but ends up in the other lane after the turn. Drives back to own lane after the turn.		
									The correct decision in the intersection.		

Bjørn André Aaslund

A Structured Approach to Autonomous Driving in Simulated Environments

NTNU
Norwegian University of
Science and Technology