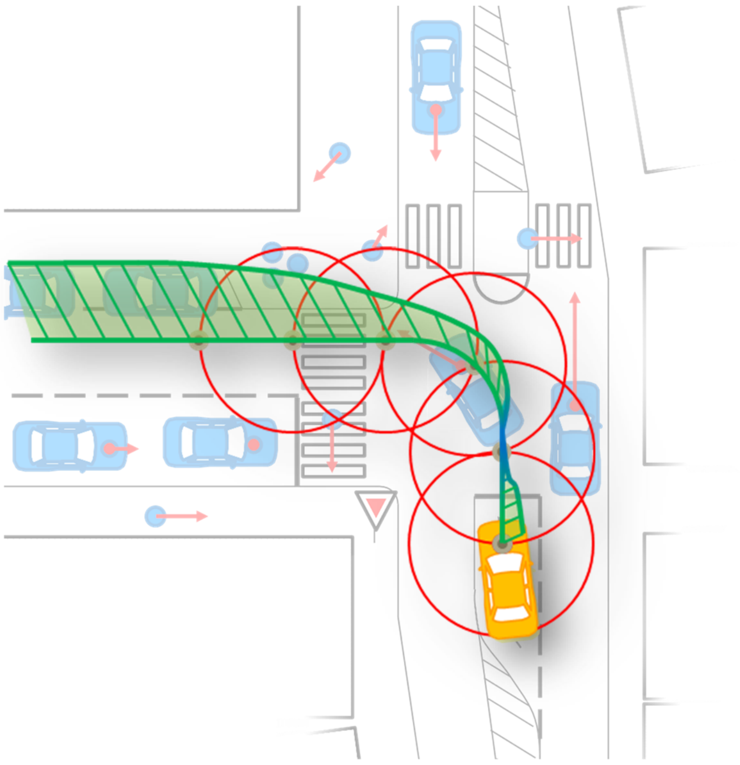
AI-based movement planning for autonomous and teleoperated vehicles including the development of a simulation environment and an intelligent agent



Master thesis submitted in fulfillment of the requirements for the degree of M. Sc.

written at the Department of Mechanical Engineering at the Technical University of Munich.

Chair	Univ.-Prof. Dr.-Ing. Markus Lienkamp Institute of Automotive Technology
Supervisor	Jean-Michael Georg, M.Sc. Institute of Automotive Technology
Written by	Thomas Nützel, B. Sc. Matr. 03640621, thomas.nuetzel@tum.de
Date of submission	15.07.2018

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Task formulation

AI-based movement planning for autonomous and teleoperated vehicles including the development of a simulation environment and an intelligent agent

Automated and teleoperated driving has recently faced increased interest in society and science. The latest developments and improvements in the fields of Artificial Intelligence and Machine Learning have resulted in new solutions that offer decisive advantages in the subject compared to conventional software solutions [1]. Movement planning is considered a major task and difficulty in autonomous driving [2]. Neural Networks offer a solution to predict the surrounding traffic and generate maneuver plans accordingly, while parallelly allowing a significant level of generalization to master unseen situations [3, p. 243]. Teleoperated driving, as the link between manual and autonomous driving, also benefits from precise movement planning. A machine computed trajectory helps the operator to compensate uncertainties caused by latency fluctuations in the mobile connection.

The following points are to be investigated by Mr. Thomas Nützel:

- Introduction to the topics of automated and teleoperated driving, as well as neural networks
- Development of the system's structure to accomplish the movement planning task
- Programming a simulation environment to generate training data
- Designing a Neural Network approach and tuning its Hyperparameters
- Training and evaluating Neural Networks
- Implementation of the algorithm into an intelligent Agent

The elaboration should document the individual work steps in a clear form. The candidate undertakes to write the master thesis independently and to state the scientific sources used by him.

The submitted work remains the property of the chair as an examination document and may only be made accessible to third parties with the consent of the chair holder.

Assignment: 15.01.2018 Submission: 15.07.2018

Prof. Dr.-Ing. M. Lienkamp Supervisor: Jean-Michael Georg, M.Sc.

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Geheimhaltungsverpflichtung

Herr: **Nützel Thomas**

Im Rahmen der Angebotserstellung und der Bearbeitung von Forschungs- und Entwicklungsverträgen erhält der Lehrstuhl für Fahrzeugtechnik der Technischen Universität München regelmäßig Zugang zu vertraulichen oder geheimen Unterlagen oder Sachverhalten industrieller Kunden, wie z.B. Technologien, heutige oder zukünftige Produkte, insbesondere Prototypen, Methoden und Verfahren, technische Spezifikationen oder auch organisatorische Sachverhalte.

Der Unterzeichner verpflichtet sich, alle derartigen Informationen und Unterlagen, die ihm während seiner Tätigkeit am Lehrstuhl für Fahrzeugtechnik zugänglich werden, strikt vertraulich zu behandeln.

Er verpflichtet sich insbesondere

- derartige Informationen betriebsintern zum Zwecke der Diskussion nur dann zu verwenden, wenn ein ihm erteilter Auftrag dies erfordert,
- keine derartigen Informationen ohne die vorherige schriftliche Zustimmung des betreffenden Kunden an Dritte weiterzuleiten,
- keine Fotografien, Zeichnungen oder sonstige Darstellungen von Prototypen oder technischen Unterlagen hierzu anzufertigen,
- auf Anforderung des Lehrstuhls für Fahrzeugtechnik oder unaufgefordert spätestens bei seinem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik alle Dokumente und Datenträger, die derartige Informationen enthalten, an Lehrstuhl für Fahrzeugtechnik zurückzugeben.

Eine besondere Sorgfalt gilt im Umgang mit digitalen Daten:

- Kein Dateiaustausch über Dropbox, Skydrive o.ä.
- Keine vertraulichen Informationen unverschlüsselt über Email versenden.
- Wenn geschäftliche Emails mit dem Handy synchronisiert werden, darf dieses nicht in die Cloud (z.B. iCloud) synchronisiert werden, da sonst die Emails auf dem Server des Anbieters liegen.
- Die Kommunikation sollte nach Möglichkeit über die TUM-Mailadresse erfolgen. Diese Emails dürfen nicht an Postfächer anderer Emailprovider (z.B.: gmail.com) weitergeleitet werden.

Die Verpflichtung zur Geheimhaltung endet nicht mit dem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik, sondern bleibt 5 Jahre nach dem Zeitpunkt des Ausscheidens in vollem Umfang bestehen.

Der Unterzeichner willigt ein, dass die Inhalte seiner Studienarbeit in darauf aufbauenden Studienarbeiten und Dissertationen mit der nötigen Kennzeichnung verwendet werden dürfen.

Datum: 15.07.2018

Unterschrift: ____________________________________

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Erklärung

Ich versichere hiermit, dass ich die von mir eingereichte Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Garching, den 15.07.2018

____

Thomas Nützel, B. Sc.

Acknowledgments

First and foremost, I would like to thank my supervisor Jean-Michael Georg for giving me the chance to investigate the enormously exciting subject of autonomous driving. Mr. Georg often supported me by providing hardware as well as usage rights and helped to find solutions to address file restrictions or other organizational issues.

Second, I would like to thank Florian Becker for his support in initializing, setting, and adapting the GPU server at the Institute (FTM) as well as the LRZ cluster.

I am also grateful to Tobias Schuster and Rene Dorsch for the interesting discussions and the resulting inspirations.

I would like to thank Ivan Rumenovic, Ali Önüt, Otello Piroddi, Christoph Ensinger, Alexander Szpotowicz, Samir Agabekov, Lars-Eric Schlereth, Dominik Heim, and Lars Michalowski for their help with proofreading.

Special thanks to Julia Nützel and Amin Maududi, who helped me with an additional linguistic revision.

Last but not least, I like to thank my friends and my family for their support during the master thesis.

Contents

Index of abbreviations

AI	Artificial Intelligence
ANN / NN	Artificial Neural Network
CNN	Convolutional Neural Network
DARPA	Defense Advanced Research Projects Agency
DIE	Integrated developer environment
DL	Deep Learning
DPU	Data processing units
FoV	Field of view
GPU	Graphical processing unit
GUI	Graphical user interface
HP	Hyperparameter
LRZ	Leibnitz Rechenzentrum
MSE	Mean square error
FTM	Institute of Automotive Technologies (
PID-control	Proportional–integral–derivative controller
ReLU	Rectified linear unit
RGB	Red-green-blue
RNN	Recurrent Neural Network
TUM	Technical University of Munich
UDP	User datagram protocol
UI	User interface

Symbols

Symbols	Unit	Description
a	m/s²	Acceleration
e_{fa}	m	Lateral error
d_{is}	m	Current distance between two vehicles
d_d	m	Desired distance between two vehicles
$J_{(x)}$	-	Cost value
$j_{acceleration}$	-	Cost value regarding vehicle’s acceleration
j_{jerk}	-	Cost value regarding vehicle’s jerk
$\mathrm{j}_{\mathrm{veloctiy}}$	-	Cost value regarding vehicle’s velocity
$\mathrm{j}_{\mathrm{offset}}$	-	Cost value regarding vehicle’s offset
k	-	Constant gain value for steering angle adaption
$prediction^{(i)}$	-	Predicted value regarding training example i
$\mathrm{true}^{(\mathrm{i})}$	-	Ground Truth of training example i
v_x	m/s	Longitudinal velocity
w_j	-	Weight in iteration j
α	-	Learning rate
δ	rad	Steering angle
θ_e	rad	Heading difference
σ	-	Standard deviation
τ_d	-	Adaption factor for the distance value
τ_v	-	Adaption factor for the velocity value
φ_i	m	Lateral offset of a vehicle from desired track at location i

1 Introduction

In recent years autonomous driving has faced a growing interest by society and industry. Many advantages are connected to this new technology.

These include a decrease in transportation efforts and mobility standards for elderly and disabled people can be achieved through it. Also higher traffic safety and a time gain for private persons and additional effective work hours for out of office employees including technical field service can be attained [1, 4, 5, 2].

In the past, most advancements in terms of vehicle technology came from within the automotive industry itself, where especially the German OEMs and their top-tier suppliers were at the forefront of developing and introducing new features that significantly improved safety and comfort of cars. This dominance has lately been threatened by the arise of new technologies such as AI, where traditional OEMs suddenly face competition from (non-automotive) tech companies that could exploit the vast opportunities (such as autonomous driving) that they offer [6, p. 710]. The disruptive nature both of the technologies and the aforementioned companies pose a serious threat to the “traditional” car industry.

Car manufacturer and their supplier network are the largest sector in Germanys industry with over 800,000 people employed [7, p. 543]. If car manufacturer not successfully challenging this subject, may result in an immense impact on German society and European economy.

Nevertheless, German car manufacturers still have a significant advantage over other competitors. They produce and sell cars for the higher-class segments. Therefore, they do not compete on a market with very restricted price levels. Customers are able and may be willing to pay an extra to gain access to autonomous driving features including costly sensors. On the downside their customer expects high quality solutions that may not be fulfilled by recent technologies and algorithms [8, p. 776].

1.1 Motivation

As already mentioned autonomous and teleoperated driving sees an increasing interest in society. Primarily because they offer some decisive advantages for industry, transportation and the customer sector.

Thus, many technical and legal obstacles are yet to overcome for this technology to enter the roads and markets. In many traffic situations, recent autonomous driving technology cannot handle the complexity of the driving situation within necessary safety regulations [1]. Often this is due to the fact that the perception and planning cannot compete with human level performance. One way to address these issues is having a person (operator) drive vehicles remotely in areas where chances are high that recent algorithms cannot handle the complexity. Combining both technologies, a solution can be offered that outperforms safety abilities of autonomous driving while sustaining many of its advantages.

1.2 Objective of this thesis

The Primary scope of this thesis is the development of an algorithm which automatically plans the focused vehicle trajectories in a public traffic environment, containing other road users and traffic regulation rules to follow.

Movement planning can be considered one of the main issues autonomous driving still lacks human driving capabilities. More precisely, movement planning within traffic scenarios in an environment range of 20 to 30 meters surrounding the autonomous vehicle. Developing a new approach to movement planning is, therefore, the scope of this work.

Automated systems that perform reasonably well even in complex and often to some extent unique traffic situations need to have a generalization effect. Meaning, they should not only operate on previously encountered examples in a strictly defined situation but also on slight variations. One concept which has already proven its strength in recent autonomous driving approaches, since they fulfill this requirement, are Neural Networks.

To generate a well performing Neural Network a significant amount of training data is required. Hence a simulation environment needs to be developed and programmed from the ground up to tackle this task. It must cover map generation and variation for data diversity reasons as well as simulation of behavior and interactions between vehicles, pedestrians and road rules including vital traffic signs and lights.

A Neural Network needs to be implemented in an overall system to receive the necessary data upfront, as well as to integrate the outputs as control parameters in order to move the car. Therefore, a concept for an intelligent hardware agent (AI) should be engineered which applies a generalizing Neural Network (NN) to plan the future movement of a vehicle in public traffic environment. The Institute of Automotive Technologies at the Technical University of Munich applies the SILAB traffic simulation solution for these testing purposes. The movement planning approach should then be extended with a suitable controller, implemented in the mentioned software SILAB.

1.3 Structure of the thesis

The thesis begins with a description of the state of the art relevant to the object of the paper which includes an introduction into autonomous and teleoperated driving, their issues, solutions and the need for movement planning. Additionally, a solution-neutral observation of Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL) is presented.

Next, the basic concept and structure of the developed movement planning algorithm are introduced. Employed network, advantages, and disadvantages of the solution approach are discussed. One of the main focuses in ML projects lays on the design of inputs and outputs of the network [9, p. 4]. Without getting into feature engineering details, an input concept relying on environment representations should be discussed. Trajectories are basing on a path which is represented by support points defined in a polar coordinate system and a velocity profile. After explaining the output, the chapter ends with some notes about vehicle control when applying the developed path planning algorithm.

Performance optimized training of Neural Networks requires a significant amount of data examples. This especially holds true when the complexity of the tasks increases, like in a traffic situation in which line detection, road following, collision avoidance and traffic regulations have to be mastered at the concurrently. To generate this large training data a simulation environment has been programmed. It allows for designing maps (Superscenarios) with various available items to choose from Superscenarios. To increase diversity in map architecture, a variation tool has been added. Afterward, a traffic simulation computes vehicle and pedestrian behavior (Subscenario). A ground truth generator saves states of selected vehicles and provides training data. Since only commodity desktop computers were available, a simple server-client solution has been programmed to scatter calculations amongst PCs. In chapter 3 a stepwise introduction to these software elements can be found, including some application advices and hints by the developer. More complex features are explained in greater detail afterward.

Explaining the training and getting deeper into network type and architecture is part of the fourth chapter. The applied roadmap for training and improving the NN is presented, including Hyperparameter selection, training phase control, and validation. For using more advanced Hyperparameter optimization, an evolutionary algorithm has been implemented. Various performance improvement towards vehicle behavior are described and discussed, including a self-written software tool supporting this process.

In the last step, the optimized NN is implemented in an intelligent agent solution for integration in a more realistic simulation environment (SILAB). For this, interfaces (DPUs) were developed to gather and transmit information to the network, which is described in the last chapter. These include an Arduino Joystick controller that provides user inputs to the network and a visualization UI design which illustrates relevant information as feedback.

Lastly, in the concluding chapters, a discussion about the thesis, a summary and potential improvements for further works regarding the presented movement planning approach is given. The described structure of this thesis is illustrated in Figure 1-1.

Basic knowledge in Machine Learning and Deep Learning is required to read and understand relevant points of this work without interruption. If relevant, remarks have been added to lead interested readers to background material.

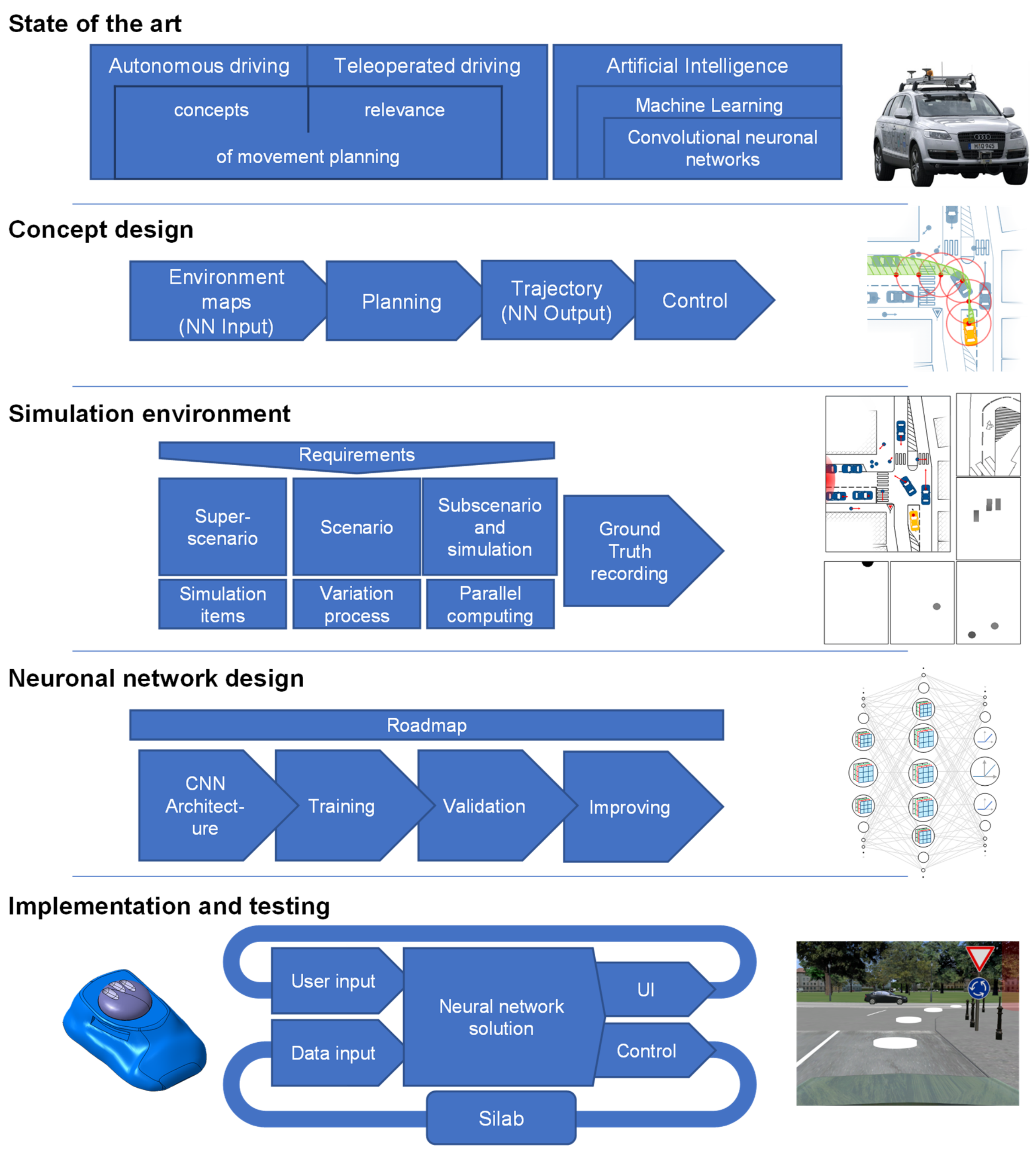


Figure 1-1: Structure of the thesis

2 State of the art

This chapter describes the state of the art considering relevant fields of the objective of this thesis. Artificial Intelligence allows new ways and concepts to solve problems that have been impossible to master before. Starting from the primary purpose of Artificial Intelligence in section 2.1.1, the focus is set on the training of an AI, called Machine Learning (section 2.1.2). Since the developed approach for movement planning consists of an image processing approach, in the last section 2.1.3 some basic information should be given about convolutional Neural Networks and their training.

In the section 2.2.1, five levels of autonomy give a first presentation of the current state of autonomous driving and its challenges. The overall driving process is then subdivided into single tasks to tighten the scope towards the topic of movement planning (section 2.2.2). A methodic concept for movement and behavior planning is presented and two diverging solutions, Mercedes Benz's conventional model optimization and NVIDIA's End-to-End approach, are explained, and discussed in greater detail in the upcoming section 2.2.3.

Teleoperated driving is introduced as an intermediate step to full autonomy. First, focusing on its basic structure and procedure (section 2.3.1). Advantages, downsides, and obstacles regarding the implementation of teleoperation are going to be discussed afterward (section 2.3.2). Teleoperated driving does not require movement planning when considering an ideal scenario with direct user input. Since this is not the case, autonomous movement planning can support the driver in various situation, as section 2.3.3 will show.

2.1 Machine Learning and Artificial Intelligence

Machine Learning (ML) and Artificial Intelligence (AI) enables automated solutions for tasks that are, until recently, primarily mastered by humans, due to its complexity, inefficiency of traditional computing solutions, and a lack of ideas on how to design and program solution approaches. Artificial Intelligence (AI) solutions divert significantly from older conventional computational solutions requiring no pre-known logic statements and deep knowledge of underlying concepts of application including their rules and implementation.

Learnable AI's are trained by Machine Learning (ML) to develop patterns for solving various tasks. While conventional programs need logics that are written in hard code ML rely heavily on data to learn from. A particular approach to ML is Deep Learning, which is capable of learn very complex model representations to perform relatively difficult tasks. An overview and relating structure of AI, ML and Deep Learning as represented in Figure 2-1.

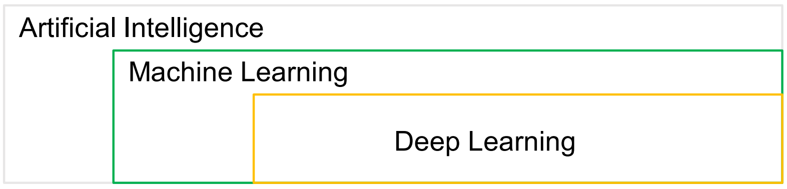


Figure 2-1 Classification of AI, ML and Deep Learning into an overall context [10]

In the next chapters, a short presentation is given to these fields. It is important to note that no detailed introduction is given in these fields, because of limitations regarding the pages of the master thesis.

2.1.1 Artificial Intelligence

Intelligence is a highly discussed term in science and public alike. Thus, the presented thesis is limited in their length and a more general discussion would massively exceed that restriction. Therefore, only solution orientated IT subjects will be discussed here.

AI can be defined as an abstract approach to find new or more efficient solutions, adopting human methods or natural procedures for problem solving [11, p. 4, 12, p. 13]. Artificial Intelligence is, therefore, an abstract concept that can be embodied and applied by an intelligent agent. An agent is a system (autonomous in this case) that serves the purpose of mastering tasks given for a host.

Particular scope in the subject of AI lays on teachable agents, that are capable of learning to improve their performance. Their intelligence is not based on a predefined logic but Machine Learning with large data sets. An autonomous vehicle can be considered as a hardware agent, as its structure illustrated in Figure 2-2. They contain sensors for the perception of the environment and actuators to change their surrounding or its own state. They observe their environment and search for other vehicles, pedestrians or objects. The inner software agent is driven by a human input like a desired destination and plan the next movements and calculate a proficient trajectory [13, p. 19]. Actors then accelerate and steer the vehicle to match the interest of the driver's input.

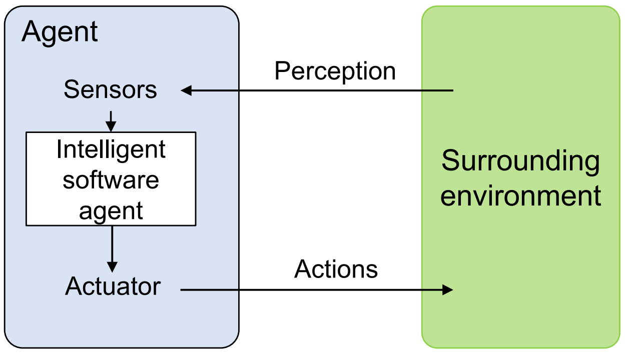


Figure 2-2: Hardware agent [13, p. 19]

Intelligent teachable agents have two modes of actions. In the learning phase, an agent's behavior is improved to perform well on the actual application phase, also known as exploit or prediction stage.

2.1.2 Machine Learning

Many issues including driving are performed reasonably well by humans but currently are challenging to master applying computer software. Machine Learning is used in favor of those conventional software when a task can be characterized by the following properties [14, p. 3].

The task is too complicated to be programmed by hand: When discussing a traffic situation on public roads, many rules are incorporated under the road traffic act which have to be followed. While each of these regulations is relatively simple to understand, the complexity of the whole system is increased with each of them.

No algorithm exists to solve the task in a satisfactory manner: When considering complex types of problems that cannot be divided into more manageable sub-tasks finding efficient general solutions becomes increasingly difficult. A concrete example of this issue might be the detection of objects in images. Thus, it might be simple for humans to describe features of the focused objects, but it’s hard to find clear sequential algorithmic solutions to implement the solution computationally.

The intelligence has to interact in a variable and therefore dynamic environment: This results in the fact that not all possible situations can be known in advance to adjust the system. Thus, hard coded and predefined rules and values will not meet the requirement of high performance of the entire product lifecycle. Training rules by Machine Learning has two advantages in this scenario. Well trained Machine Learning models generalize well on unseen data, meaning that not any possible situation must be given in advance [3, p. 243]. Nevertheless, similarity is a prerequisite. Furthermore, reinforcement learners like Q-learning algorithms are capable of learning while they are used, by collecting feedback and continuingly improving their inner models.

Machine Learning, therefore, brings decisive advantages when considering an application in autonomous driving. A disadvantage that should not be kept unmentioned is the need for additional safety measures. ML models are good at generalizing and solving similar not seen examples but tend to unpredictable reaction to dissimilar data. Additionally, it is difficult to find failures inside a Machine Learning model since it can be considered as an abstract or black box representation. Nevertheless, Machine Learning should be kept in mind, not just because of its previous success in autonomous driving.

Machine Learning itself can be subdivided into two main types. Supervised and unsupervised learning. First requires labeled training data that contains an input of an operation linked with its correct result (Ground Truth) [15, p. 4, 16, p. 55]. A supervised learner then finds rules and operations behind the set of input and result to predict further unseen tasks, similar to those it was trained on. [17, p. 35] Unsupervised learning on the other hand only requires unsolved data to find patterns (dimensional reduction) or clusters (cluster analysis) within the given dataset [16, p. 55]. A third type which often is considered as unsupervised learning but distinguishing from it in many subjects is reinforcement learning. Here an algorithm improves continuously by gathering feedback relating to its action in a complex environment [17, p. 12-13].

When training a network with a pre-generated trajectory that serves as Ground Truth, the focus for further explanations is set on supervised learning. Two basic types of possible conclusions exist for supervised learning. When using a classification approach the algorithm searches for patterns to assign inputs to a class as output [18, p. 112]. For instance, an object can be classified as a person, car, truck or traffic sign. Regression, on the other hand, allocates a value to an input [17, p. 35]. An image of a vehicle's surrounding can be transformed to a steering angle and velocity. As the example implies, trajectories containing yaw rates and velocity profile are numeric values that are predicted with regression models. Several regression algorithms are available to generate a model, including linear regression and decision trees for simple predictions. Support vector regression and Artificial Neural Networks (ANN or short NN) are applied for more complex and non-linear problems. The full table containing explained ML algorithms is illustrated in Figure 2-3.

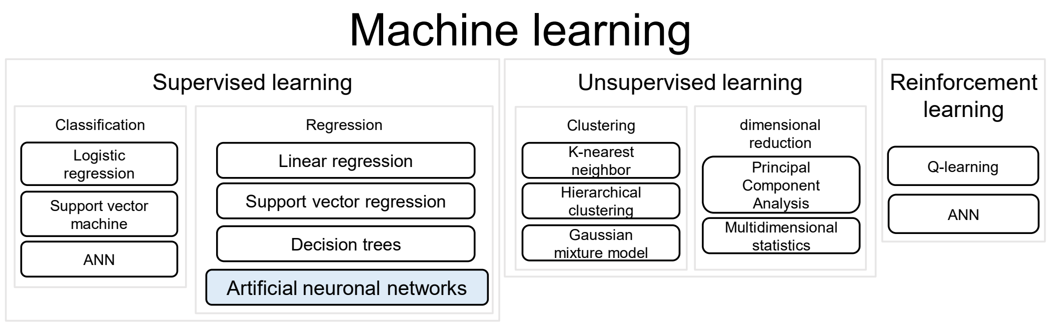


Figure 2-3 Machine Learning algorithms [19]

2.1.3 Neural Networks and Deep Learning

As mentioned above, Neural Networks are primarily used as supervised Machine Learning tools. Additionally, it should be stated that NN can be applied in reinforcement learning as well. The *Audi traffic jam pilot* is based on a NN to predict the value of upcoming actions [20]. Since the disproportionately reduced usage of these Networks and the restrictions in length of the master thesis, the subject will not be covered in greater detail. This work is primarily focusing on NNs that are trained on prelabeled examples and are therefore assigned to supervised learning.

Nevertheless, there are some differences between supervised learning in Machine Learning and Deep Learning. In ML features are required, while the NN will also work with raw data and is extracting relevant features by its own. This difference can be illustrated considering the following example: Based on camera footage, a software should decide, whether a person is a minor or an adult. To use a Machine Learning algorithm a process called feature engineering is necessary to predefine properties and their range to assure a correct prediction stochastically. An obvious feature to start with might be the body height of the person. Having some examples of people with their height (feature) and the information of different properties are needed to obtain a more accurate assumption. So, the list of features must be extended. Choosing wrong features can even impair the performance of ML algorithms in some cases. In a next step, the Machine Learning algorithm is searching for ranges to classify a person as a minor or an adult by creating rules itself. Prefixed to these steps of estimating the age, it might be necessary to identify a person on the video stream, which adds additional complexity to the task.

In contrast to this, NN exemplary for identification of persons on a video stream requires simply many pictures or video recordings of persons labeled with the classes minor or adult as Ground Truth. This advantage of not requiring preset features causes additional work for the NN. Since the network finds patterns on more complex raw data, the number of examples which are necessary for training a reasonably well-performing NN massively exceeds those from feature requiring ML algorithms.

The overall application of these Networks is subdivided in different phases. Starting with the learning phase, the NN is trained on examples containing an input and a predefined labeled output [21, p. 56]. The learning process itself is divided into two steps. As mentioned, this chapter is considered as a refreshment, not as a detailed introduction to the subject. In the first step an example will be forward propagated through the net. Each node of a net is provided with the accumulation of nodes or filter parameters of the previous layer including a bias value [22, p. 24]. This sum is then activating a neuron which is characterized by one of different activation functions like the ReLU function (others can be found in [21, p. 65-70, 23, p. 13-16]) and a prediction is calculated for the output. A cost function is set up to calculate a difference measure between predicted and actual value, which has to been predefined before a training session begins. With Formula (2-1) a very common mean square error (MSE) is presented (others can be found in [21, p. 72-75]).

$$J(w) = \frac{1}{n}\sum_{i}^{n}\left(true^{(i)} - prediction^{(i)}\right)^2 \tag{2-1}$$

The cost function is then optimized by an optimizer like Gradient decent or Adams. The update process of gradient descent is illustrated in Figure 2-4. With each step towards the minimum, the slope decreases. Therefore, the update process slows down.

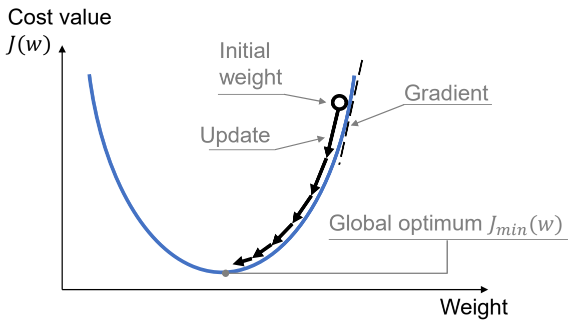


Figure 2-4: updating error function with gradient decent [13]

Backward propagation then updates each weight and bias gradually back to the input layer, as illustrated in Formula (2-2) [24, p. 157]. A learning rate parameter is defined to adjust “update strength” within one step. Decreasing can lead to slow down the learning process, increasing may result in oscillating of the gradient and dismissing saturating at the optimum.

$$w_{j,new} = -\alpha\frac{\delta J}{\delta w_j} + w_j \tag{2-2}$$

ANNs are an umbrella term and combines different types of Neural Networks. These types differentiate vastly by their architecture (components and structure), use cases and goals.

Feed Forward or Dense NN: Starting with the Feed Forward Networks it has to be mentioned that they consist of the fundamental elements and are therefore present in all other ANNs to some extent.

Convolutional NN (CNN): Instead of nodes and single weights between nodes, they use image filters as learnable parameters and pooling layers for information reduction. CNNs are used for image processing, detection, and classification [23, p. 99].

Recurrent NN (RNN): Are used, when either the length of an input (signal) is not fix in foresight or a prediction requires multiple data points in a timeline. Therefore, it is often built in software for audio processing, speech recognition or translation, since no fixed sized in- or output is required [21, p. 159]. Another reason to use RNN's is the lack of information transfer within subsequent Iterations in the learning and exploiting phases. While normal ANNs are trained on one data point after another, Recurrent networks are capable of processing a sequence of information as an input.

In Deep Learning two types of model variables are distinguished that influence prediction performance. Parameters are optimized or "learned" during the training phase. A set of Hyperparameters (HP) on the other hand defines the architecture and different operational adjustments of the focusing NN. The HP including their parametrization characterize a trained Network. The choice of matching HPs has a vast impact on the possible performance of the Network. Therefore, it is crucial for the Machine Learning projects to choose a starting set of HPs carefully. Usually, the starting set of HPs is not optimal. Thus, an essential step in realizing an AI-Project is the iterative HP tuning process.

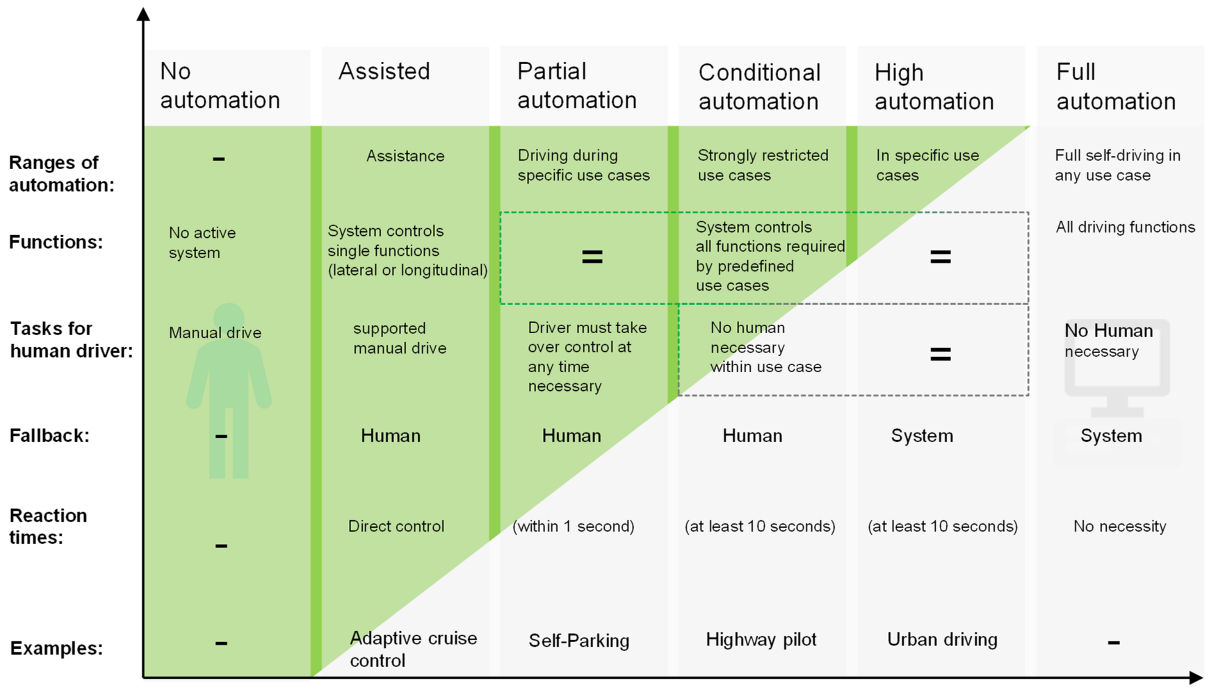
2.2 Autonomous driving

As shown in the introduction, autonomous driving sees rising interest in science, industry, and society. Despite massive efforts, no satisfactory implementation of autonomous driving exists to this day. This chapter should give the reader a first introduction to autonomous driving and a description of its obstacles.

The chapter starts with a subdivision of autonomous driving and description of the current state of the art (section 2.2.1). Before digging deeper into various concepts of autonomous driving, it is necessary to understand which tasks a computer has to handle when it comes to autonomous driving. The 3-Layer-model gives a short overview and a clear structure about these initial driving tasks from a human perspective (section 2.2.2). Lastly, some existing concepts focusing on the complex tasks of automated vehicles and movement planning are introduced (section 2.2.3).

2.2.1 Levels of autonomy in automated driving

Various classifications exist for the automation of vehicles and especially autonomous cars [1, 25, 26, p. 15]. The divisions often are similar, but they exceed each other by different aspects. Instead of describing a single one, a combination has been worked out that combines the most relevant elements from multiple classifications. An overview is given in Figure 2-5.



	No automation	Assisted	Partial automation	Conditional automation	High automation	Full automation
Ranges of automation:	-	Assistance	Driving during specific use cases	Strongly restricted use cases	In specific use cases	Full self-driving in any use case
Functions:	No active system	System controls single functions (lateral or longitudinal)	=	System controls all functions required by predefined use cases	=	All driving functions
Tasks for human driver:	Manual drive	supported manual drive	Driver must take over control at any time necessary	No human necessary within use case	=	No Human necessary
Fallback:	-	Human	Human	Human	System	System
Reaction times:	-	Direct control	(within 1 second)	(at least 10 seconds)	(at least 10 seconds)	No necessity
Examples:	-	Adaptive cruise control	Self-Parking	Highway pilot	Urban driving	-

Figure 2-5: Overview of vehicle automation classification

As it is the case with most diversifications, a six-member classification was chosen. Starting from no automation, the driver relies on its own skills and capabilities with no system active while driving. Therefore, the attention is focused on traffic situation continuously.

This stage has been exceeded soon when first stabilization systems came on the market in the 70s. Since then assisted driving was possibly leading to increasing levels of comfort and safety. In this case, a vehicle usually controls a single subtask, including lateral or longitudinal guidance. Often no additional intelligence is part of these solutions, mathematical formulas exist to describe and solve related tasks. Since the systems can be considered as a support, the driver it is still the active part of vehicle maneuvering. Examples of these systems are the cruise control or lane keeping assist systems.

Partial automation takes control over the vehicle if a situation or tasks occurs that are covered by the solution's use-case and capabilities. For accomplishing these tasks, multiple driving functions could be required which are handled by the system. In case of an emergency, the human driver is still responsible for taking control of the vehicle within a second, meaning that attention is still focused on the road. Automated parking systems allow maneuvering the vehicle while the driver might stay outside of the vehicle and must stop the maneuver by pressing a button on the key before a collision occurs. Another system that is considered as partial automated is the autopilot feature in a Tesla vehicle.

A system which is classified as conditional automated does not require cognitive human observation during fulfillment of tasks. Instead, the driver can focus his attention on other activities like media consumption. When a system reaches its boundaries, the driver is informed and has ten seconds left to perceive the traffic situation and plan next required steps. Therefore, as a fallback system, the driver must be capable of taking over the control over the vehicle if use case conditions are not met anymore. *Audi* released the traffic jam autopilot in late 2017 which is the first conditional automated system on the market. The system boundaries are stringent, including a speed limit of 50 km/h limited to a highway traffic jam scenario [20].

The main differences between conditional and high automation lay in the fact that the applied algorithms can bring the vehicle in a safe state when use case conditions are not met anymore and the driver is not capable of taking over control. Save states imply maneuvering the vehicle to the next pull-of bay or breakdown line, without disturbing or interrupting the traffic flow. No system is obtainable on the consumer market currently.

Full autonomy then does not require any human driver. All potential traffic use cases are handled by the system on at least human-level performance.

As mentioned, the first conditional automated systems are currently entering the market with many obstacles to overcome in order to reach full autonomy as presented above. Technical and legal issues including responsibility for the vehicle's behavior are significant reasons for the current state of solutions on the market [1, 27, p. 6-10]. To get a more profound understanding of the obstacles of the technical development of autonomous driving the next section gives an overview of driving tasks that are required to maneuver a vehicle to the desired location safely.

2.2.2 Separation of driving tasks

The widely known *3-layer-model of driving tasks* gives a good representation and a clear subdivision of driving as a whole. The model divides the overall driving task in three separate actions starting from the navigation on an abstract map like level to movement planning and vehicle stabilization as a very physical formulized task. An overview of these tasks is illustrated in Figure 2-6.[28]

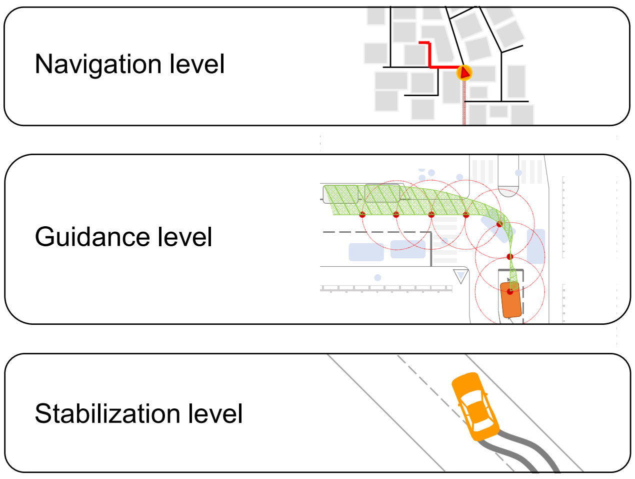


Figure 2-6 3-layer-model of driving tasks

Starting with the navigation level, the optimal route is calculated using a street network, presented as a map for humans. This includes avoiding wrong-way driving in one-way streets or respecting route restrictions like height or weight restrictions at tunnels or bridges.

Skipping the second level, task three, also considered as stabilization level, manages vehicle dynamics on the boundaries of driving. Avoiding over- and understeer as well as forcing predictable vehicle behavior on rough ground and when braking is considered as relevant responsibilities covered by this level.

The second level can be considered as the most evident and cognitive intensive task when driving is subject in ordinary discusses. A vehicle is set in a specific traffic situation having information about its surrounding in a 20 to 30-meter range. Additionally, many traffic regulation rules have to be considered on this level. Third-party vehicles and pedestrian's behavior become relevant and need to be predicted in order to master the situation successfully. The task relies on constraints and restrictions of the two other levels. Considering all this information, a movement has to be planned, including a steering angle (lateral) and acceleration (longitudinal).

All three tasks require different skill sets. While navigation involves information about current rough location, street network and destination it is mainly knowledge driven. Routing requires relatively simple algorithms like A*, Dijkstra, etc. [29, p. 39] in order to solve fundamental discrete optimization problems.

Track guidance considers routing on a more detailed level. Perception prediction and movement planning are relevant tasks. Furthermore, it can be viewed as a time critical task. Fast reaction times are required to interact with other road users especially in situations with higher differences in velocity. In several scenarios communication is indispensable. Cases in which a collision might be occurring must be detected and prevented.

Stabilization requires microcontrollers allowing fast refresh rates and vehicle dynamics sensors (like revolution counters attached to the wheel, etc.).

When taking all three differential driving tasks into consideration, one can clearly determine which functions a machine can already perform, and which require additional effort in order to find suitable solutions. Basically, solutions for navigating and stabilization already exist and outperform human capabilities regarding calculation speed and reaction times. One major obstacle that remains is track guidance since it requires various skills that are difficult to describe as a generalize computational solution. The scope of this thesis is to develop a new movement planning system that should integrate all mentioned information concerning road users and various road regulations to calculate a sufficient movement for the focused autonomous vehicle in a traffic situation. Two related examples of path planning solutions should be introduced in the next chapter before describing the own design approach.

2.2.3 Approaches to movement planning in autonomous driving

Since the topic of movement planning is very complicated, many solutions exist, especially through 15 years of DARPA challenges [30–35]. Furthermore, each explanation would considerably take up several pages. Hence their description exceeds beyond the scope of an entering chapter of a master thesis. Therefore, only two very diversified methods should be explained briefly to point out the range of possible solutions to this very complex subject. The first, Mercedes Benz's, Berta project, has been developed before and the second, NVIDIA's End-to-End approach, after the rising interest in Deep Learning. Both being solutions reaching for level five as fully automated driving.

Starting with Mercedes Benz's approach known as Mercedes Benz Berta, which was developed and tested in 2013, years before Deep Learning was considered capable of solving those complex issues. In principle, the problem has been addressed by optimizing geometric problems containing complex nonlinear functions [36, p. 79], which are represented by Formula (2-3). Here, the equation describes an integral reaching over the next seconds, containing various cost functions in it. Cost functions comprised the offset from the desired trajectory (following the middle of the road) and velocity (speed limitations) as well as the size of the comfort relevant values acceleration, jerk, and yaw rate [36, p. 89].

$$J(x) = \int_{t_0}^{t_0+T} (j_{offset}(t) + j_{velocity}(t) + j_{acceleration}(t) + j_{jerk}(t) + j_{yaw\,rate}(t))\,dt \qquad (2\text{-}3)$$

To calculate the ideal trajectory path, high definition maps were captured with a vehicle's sensors before the test drive. The optimal trajectory has been recorded via satellite navigation. A localization step was then acquired to calculate vehicle's position to determine the potential offset of a trajectory. Apart from this precalculated path, third-party vehicles amongst others have been taken into account. They manipulate the form of the trajectories by their appearance on predicted corridor areas (Figure 2-7) at a predetermined time. The trajectory then is reshaped to avoid colliding with them. [36, p. 87]

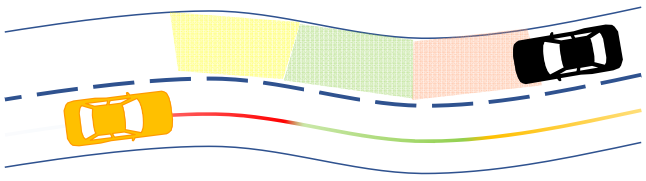


Figure 2-7: Prediction of vehicles position [36, p. 87]

Optimizing the path of the ego-vehicle then means minimizing the cost function of the whole integral using differential equations.

NVIDIA developed a very different approach to autonomous driving, applying a Neural Network on the raw input sensors to directly calculate steering and acceleration output. Instead of dividing perception planning and controlling all steps are combined and calculated by a single Neural Network. An image stream is processed by a CNN. Direct steering angle and pedal input are exported and sent to the vehicle control system directly. The computation unit contains five convolutional and four dense layers including 250,000 parameters in total. Instead of pooling layers a stride was applied to reduce the size of inner layers accordingly. [37]

2.3 Teleoperated driving

Due to poor performance compared to human driver many traffic situations (e.g. city traffic) cannot safely be handled by algorithms. Other settings (e.g. Highway traffic) with less complexity do not represent a problem for autonomous driving systems. While autonomous driving got stuck facing issues mainly focusing on track guidance level containing perception and movement planning, teleoperated driving uses a human to solve this problem with a more conservative approach.

Teleoperation contains a fix structure that is explained in section 2.3.1. The implementation of teleoperation incudes several advantages and disadvantages compared to autonomous driving that are explained in section 2.3.2. Since human are reasonably well with movement planning, in section 2.3.3 it should be described how an algorithm can support the operator to maneuver his vehicle more precisely.

2.3.1 Structure of teleoperated driving concepts

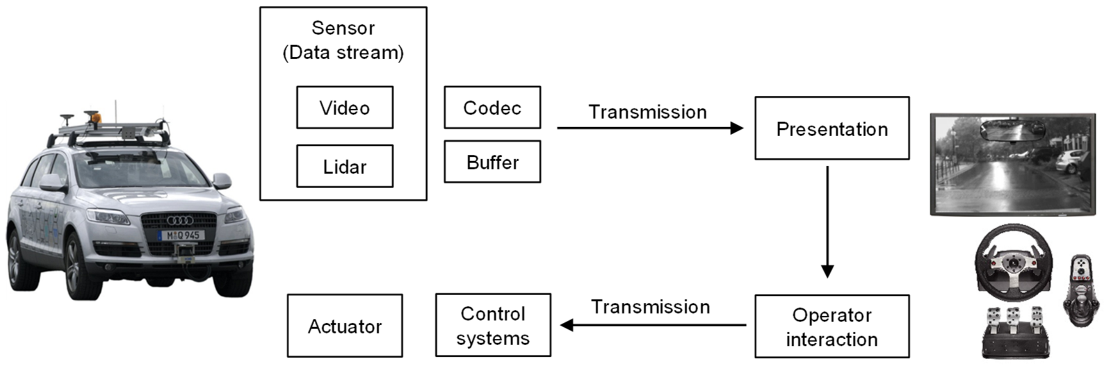


Figure 2-8 Teleoperated driving schematic

An Audi Q7 is in use for experimental purposes at the teleoperated driving team at TUM. The vehicle is currently equipped with a Lidar, short and mid-range Radar as well as several cameras pointing to all directions. Camera sensor data is transformed into a byte-stream by applying a H.264 AVC video codec [38]. This stream is then sent via the mobile network to a data center where it gets reassembled by the codec and buffered to balance lags and then presented to a driver (operator) [39]. A workspace of an operator contains familiar elements strongly inspired by a normal vehicle's cockpit. This includes a steering wheel, a gear shift lever, pedals, a dashboard and multiple displays to show the surroundings of the vehicle. Special camera streams can be shown on the screen when necessary (rear view camera when parking). Inputs from the driver will be sent directly to the vehicle via mobile communication, including steering wheel angle and pedal state. A control system adjusts actor settings and states to realize the desired movement [40].

2.3.2 Chances and difficulties

Teleoperated driving offers many opportunities for improvements in transportation. Starting with an efficiency gain in the transportation sector by requiring less workforce when combining teleoperated driving with technologies as platooning or autonomous driving on more easily manageable highways. The utilization of vehicles in a car-sharing environment can be increased by driving them to customer locations and therefore allow easy access and fast availability[39]. Another possible use case can be found in the agricultural sector. Tractors are already capable of autonomously executing fieldwork [41]. When driving on public roads, velocity is largely restricted or limited because of technical issues. Furthermore, velocities below 30 km/h already have been proven to be manageable for an operator [38].

Summing up, advantages are given in scenarios where a driver maneuvering a single vehicle is economically or practically not suitable. A single driver located in a central operator workspace can maneuver several vehicles sequentially.

The major difficulty when implementing a teleoperated driving approach is data transmission from the vehicle to the operator. What comes in mind first is the delay between vehicles perception and the video stream that is presented to the driver [39]. This is directly coupled with the data transmission rate and speed. New and faster standards are arriving with 5G telecommunication standards, though.

The most significant portion of the delay comes from buffering the image stream. These allow presenting a continuous video stream and balance lag spikes [39]. Humans determine speed in videos by comparing the changes which result from the movement of the focused object. This skill can be comprised when the pictures in a video are presented not within the codec specified time interval. A proper feeling for the speed of objects requires right the timing of images, which can be accomplished by saving them for a specific delay time to ensure that transitioned images are available at that point. If this is not done lags can provoke collisions because of false speed interpretation [39].

Another critical point that might come while introducing teleoperated driving outside of experimental environment is the critical reflection of the broad public. When an accident occurs with a teleoperated vehicle as a participant, no first aid interactions and securing of the scene of an accident can be applied by the driver for obvious reasons.

Additionally, it can be argued that while a machine may overperform human skillsets and therefore is capable of decreasing the likelihood of accidents, teleoperated driving relies on human perception. Thus, teleoperated driving will although stick with its downsides in addition to the risk of broadcasting issues like delays and lags. Therefore, it will always underperform compared to a human driver in an analog driving situation that does not suffer from delay.

2.3.3 Movement planning in a teleoperated driving concept

After considering the advantages of teleoperated driving, one might ask why movement and trajectory planning is a relevant subject in this field. Especially, since a human already masters this task with sufficient performance.

Considering this delay, it is challenging to precisely maneuver the vehicle to a specific location or following an optimal path. Since the driver always is presented with a past state, direct controlling will lead to oversteer due to the need for counterbalancing issues. This results in the fact that speed was limited to 30 km/h during teleoperated test drives within the university project [38].

Not to mention the additional crucial task of predicting vehicle’s and pedestrian’s behavior. Since this also includes reacting to the teleoperated vehicle which perception and actions are delayed because of transition time. Summing up, precise movement and prediction is considered very difficult for humans when a delay is added to informational input and vehicle output [38]. One way to encounter this problem is to reduce necessary preciseness and time criticality of operator interaction by calculating a detailed trajectory on a computational unit within the vehicle, which satisfies more rough constraints given by the operator.

3 Concept creation of a movement planning algorithm

After giving an overview of the subjects of autonomous and teleoperated driving in this chapter the developed concept of movement planning should be discussed. Before getting into details on the structure of the new approach, few key design decisions should be stressed out. Therefore, section 3.1 will chapter give a first implication about the critical elements of the developed algorithm. Furthermore, in section 3.2 it should be compared with the presented solutions of NVIDIA and Mercedes Benz.

3.1 Overall combined structure

As mentioned, in chapter 2.2.2 the performing tasks in vehicle control can be subdivided into three elements, containing perception, planning, and control. Besides, sensors are required as an input for the perception and actors are driven by control interfaces to execute movement commands. To allow a more detailed view of the developed solution the elements perception and control can be repeatedly subdivided into detection and classification, perception as well as trajectory and control [5, p. 3]. Figure 3-1 illustrates the mentioned elements, relevant parts that are focused in this thesis are shaded in blue.

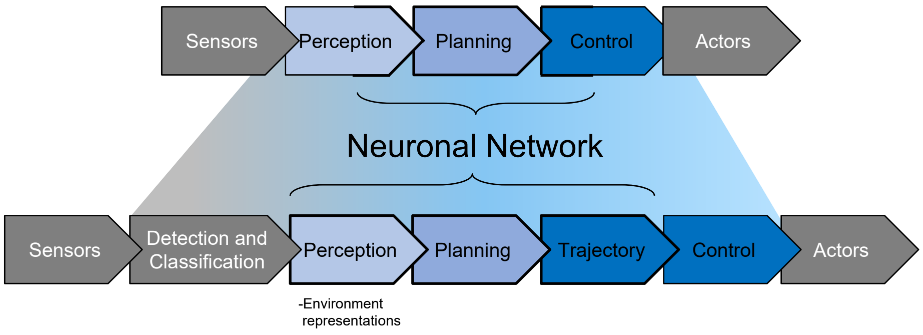


Figure 3-1 Structure of movement planning AI

Sensors: Raw input data is provided by vehicles sensors including Lidar, Radar, and camera streams as well as ultrasonic sensors. First noise canceling, and preprocessing is subordinated to this element. Sensor fusion can be applied when attached tasks require multiple sensor inputs.

Detection and Classification: blank sensory data includes raw RGB Pixels, time of flight data and orientation etc., which needs further processing to detect and classify objects an intelligent agent has to consider when planning its movements.

Perception: After detection and classification objects a meaning must be assigned. Perception in this thesis contains the creation of an understanding of the underlying concept of relevant elements in a road traffic scenario. Exemplary, this includes the meaning and effect of traffic signs and lights, road markings, vehicles, pedestrians and their behavior. Vehicles are amenable to the law of dynamics. This implies various limitation to their movement including that they will not move sideways, while pedestrians have the potential to do so.

Planning: Applying this knowledge to the required task of movement planning a system has to transform all information into a plan of action. Planning should also consider the effect of its output to the own vehicle (stable drive required) as well as feedback reaction resulting from surrounding traffic.

Trajectory: In this master thesis a trajectory was selected as a movement plan representation, containing a geographical path and a velocity profile. The trajectory must fulfill all constraints given by the planning algorithm and is, therefore, an optimized solution of the underlying movement planning issue.

Control: After optimizing a trajectory a control interface must be provided to translate information to executable orders for the actuators. An actuator cannot handle a path. However, it requests direct details on steering angle rotation, brake, and injection pump pressure. To reduce complexity, the controlled parameters will be a steering angle, accelerator, and braking pedal position.

Actors: In a final step, controlled parameters like steering angle and pedal state are transformed into actual movement using implemented actors of the vehicle.

Neither Sensors nor Actors are within the scope of this work and are therefore considered as prerequisites. Other study works covers detection and classification at the Institute of Automotive Technologies at TUM. Thus, these topics are excluded as well.

As been mentioned the planning part will be achieved using a Convolutional Neural Network. When designing a CNN the primal step is to define a suitable input and output of the bounding layers. This adds additional modularity to the approach since CNNs are capable of dealing with a variable amount of input channels that is predefined before training. When it is found relevant, an additional channel can be added to the network as will be shown later in this chapter.

Before getting into detail about the CNN concept, it should first be stressed out why Neural Networks have advantages to conventional software solutions (Mercedes Benz's Berta project) in path-finding and movement planning scenarios considering road traffic situations.

Neural Networks are expected to show good results within movement planning mainly due to their good generalization effect. When training on similar examples, new unseen situations are handled analogically. Imagine that no traffic scenario, street course, or intersection appears alike, but with tiny variation, good results can be expected [3, p. 243, 37]. Additionally, no clear scope exists on how to program solutions for any possible scenario, especially when forecasting on the behavior of road users might be required. A software which could be capable of achieving this would contain a very complex structure and is hard to maintain or extend. Conventional computer programs therefore probably are not the first choice solving this existing complexity in traffic environments, which is why further investigations focused on NNs.

When designing a concept for movement planning its components containing an input, CNN, trajectory output, and vehicle control needs some further consideration. In the next paragraphs, the principle workflow and a description of all components are presented.

3.1.1 Input and perception of the Neural Network approach

After enumerating several pros and cons of using a NN to address movement planning issue, a more detailed explanation about the embedded solution should be given. When using environment representations as an information carrier, it is appropriate to use a Convolution NN as an underlying architecture. They either can handle classification, as well as in this subject required regression tasks. CNNs are conveniently acquired for computer vision tasks like detection or segmentation. Generally, the input of a CNN represents colored images with separate red, green and blue (RGB) channels. CNNs can input a freely selectable number of channels that are fix in size (width and height). In this approach instead of passing images containing multiple color channels, different map types, including a road guidance, vehicle, pedestrian, traffic sign and target map are provided. These maps are further referred to as environment representations. An illustrating example is shown in Figure 3-2.

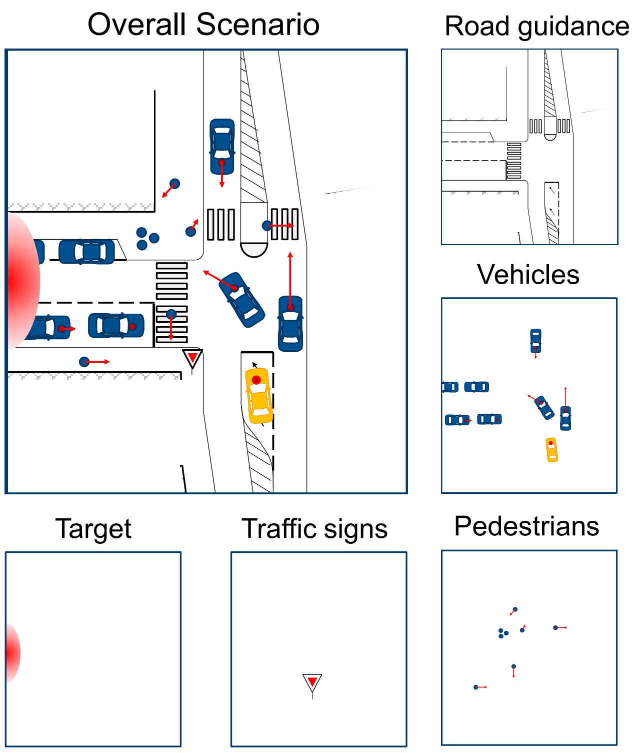


Figure 3-2: Input maps for the CNN

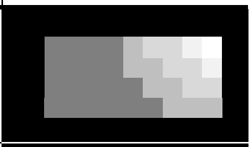
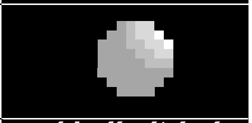
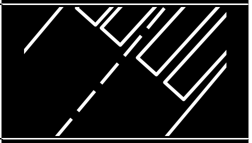
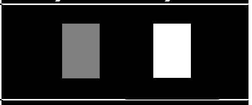
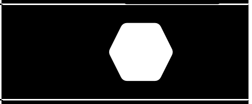
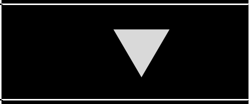
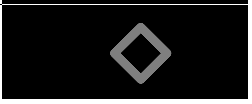
It is arguable that a single environment representation containing all relevant elements might result in the same performance. Separating information about vehicles, line markings, and traffic signs requires additional processing steps and computing power.

During the start of this work, no fundamentals on the effect of information separation using feature exclusive channels were available. NNs can be considered a black box models with little insights including cryptic feature maps resulting from neuron activation patterns [37]. Therefore, there does not exist a scientific argument against the upper claim. Nevertheless, this approach has been chosen because of two reasons. First, input maps may come from different sources. Road markings, for example, can be provided by detailed maps, while vehicle information is generated at runtime using various sensors.

The second reason can be derived from conventional RGB image processing. The idea behind separating ground colors is to allow the system to work with individual color values reaching from zero to 255 and therefore increasing accuracy by finding and distinguishing relevant features than using a grayscale image. Translated to the presented problem a separation of channels can increase the width and depth of information which can be provided to the network.

This allows adding information like velocity and heading of surrounding cars or states of nearby traffic lights. The shape of pedestrians might be increased and standardized to dismiss discrimination and reduce complexity. All abstraction and information enhancement transformations in environment representations used by the CNNs approach are listed in Table 3-1.

Table 3-1: Elements including their representations on the environment representations

Elements	Abstraction of shape	Information enhancement by shading	Appearance in environment representation
Vehicles	Bounding boxes of vehicle	Velocity and turning direction (see although chapter 4.4.5)	
Pedestrians	Bounding boxes with a standardized shape	Velocity and turning direction	
Road marks	Non	Discrete Boolean appearance (0≙ no mark, 1≙ mark)	
Traffic lights	Standardized shape (10x4 pixels)	Dark value = green Bright value = red (stop)	
Traffic sign: Stop	Standardized shape (rhombus)	Bright value	
Traffic sign: Yield sign	Standardized shape (triangle)	Medium bright value	
Traffic sign: Priority	Standardized shape (rhombus)	Dark value	

3.1.2 Planning considering a Neural Network approach

In this project, the intelligent agent responsible for planning actions is trained by Deep Learning to predict future environment states and react accordingly.

A Neural Network is considered as a black box since little insight is available. Networks in its basic form only consist of learnable weights and biases. At the beginning stage of development, an idea was discussed to engineer some connections of starting nodes manually to force a wished behavior. For instance, multiple filters can be applied to only a combination of channels like traffic lights, signs and current ego vehicle velocity to train the behavior of decelerating when a traffic light is switched to red, or a stop sign appears. This idea was abandoned quickly since Neural Networks learn these weights thoroughly on their own while they search for logics or patterns that match data best. If the best logics require some filters where certain inputs do not have an influence on during the training phase, their weights would be learned as zero automatically.

The architecture is part of the Hyperparameters of the network. These differentiate from simple network parameters in the fact that they must be predefined before starting the learning process, since they are not learned like weights and bias values automatically. Settings of Hyperparameter have significant effects on network performance. Since network training and Hyperparameter optimization are a large and complex subject, they need special consideration and therefore are excluded from this section but are presented in chapter 5.

A subject that has a significant effect on the performance of the resulting agent is the choice of a sufficient output representation, both for the full AI solution as well as the learning capabilities of the network. The decision made in favor of a trajectory design as an output instead of a single angle as it is used in the NVIDIA End-to-End approach. The reason for choosing a trajectory has resulted from the idea that training a trajectory includes training on a more long-range focused solution as a single steering angle does. That becomes relevant for two reasons.

First, it might increase stability, since the error function corrects the full trajectory. The learning process does not optimize a single value and therefore concentrates on the next milliseconds/meters only, but on a more extended range, the trajectory reaches into the future. To master this, a higher-level forecasting performance is necessary to predict and react to the future behavior of vehicles and pedestrians. Focusing on the network, the effect might be comprehensible, if considering the procedure of backpropagation. When having several sequential output values (multiple velocities) for one information (velocity), one single error value will determine a single update rate for the network. For each output, a single error will be calculated which then will be backpropagated. In the penultimate layer, the learning effect only relates to the connected output. Beginning with the next steps, one error has an impact on all nodes and results in updating to find a pattern that matches not only the very next velocity but the whole velocity profile. These updates then also affect the forward propagation and other outputs as well.

A second advantage of predicting a trajectory is a reduced time-criticality compared to predicting a single angle and pedal state. If implementing a controller, interfering lags can be counterbalanced by following the calculated trajectory.

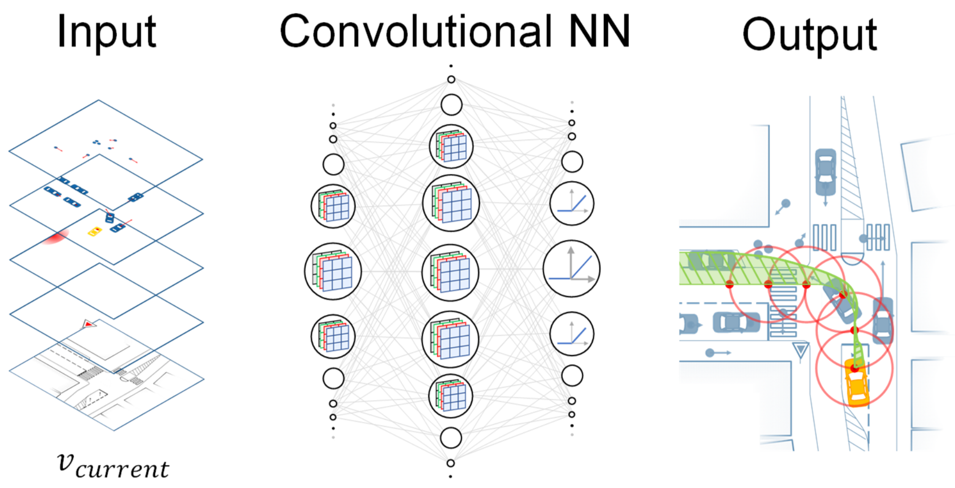


Figure 3-3: CNN with input and output

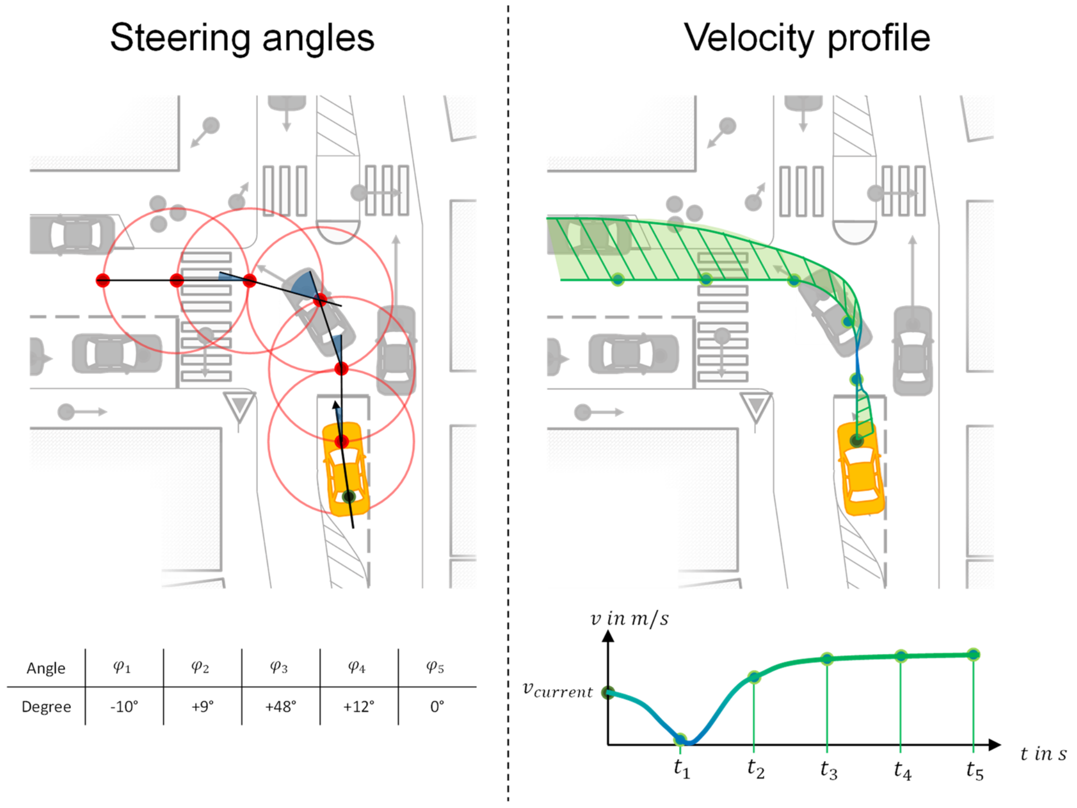
3.1.3 Trajectory as the output of the Neural Network

A trajectory is characterized by a combination of a path and velocity (or acceleration) profile, projected into the future. Both elements offer various types of possible definitions. A path can be described as an analytic function or its supporting points. Defining a path with a polynomial function seems to be the most elegant way since the number of information is limited to the number of parameters required to meet the mathematical constraints. When using a polynomial function with a degree of four requires only five parameters that must be optimized to find the best path. When using a regular one-dimensional function, the path is restricted in its directions. For instance, a U-turn, reversing or sharp S-turns cannot be described because of mathematical illegalities. When using the center point of the vehicle as the origin of the coordinates and the heading vector as the ordinate axis, a U-turn would result in a vertical line section and multiple *x*-values for a single *y*-value. A two-dimensional function can solve this issue, having two separate polynomial equations connected by a control variable. A problem that becomes relevant when applying polynomial interpolation is the risk of increasing oscillation and overshooting, if using multiple support points, as it has been described by Runge's phenomenon [42].

A favored second approach is to apply a set of support points directly for path representation. Interpolating these by applying cubic spline interpolation characterizes the exact profile. Since splines are defined by piecewise interpolation sections only considering a limited and fix number of points, they will not be restricted by Runge's phenomenon. However, for each support point, two outputs must be learned and predicted by the NN. Depending on the constraints of the polynomial and spline, twice as many information are necessary to represent a similar graph. To reduce this additional complexity, support points are not defined in a Cartesian but in a Polar coordinate system, with predefining a fix radius. This leaves the angle as learnable NN output. A four-meter range has been selected to specify the radius.

The lateral velocity profile of the trajectory has been defined by a spline as well. Instead of using two-dimensional interpolations a single one serves the purpose. Velocity is related to the time axis, thus making it independent of trajectory's path. Each support point is separated by a fixed time interval (200 ms) to reduce complexity, similar as mentioned before.

Within this thesis, only positive velocities are considered to keep the problem manageable. Each trajectory element is characterized by five support points, adding up to a 20 m range for the path and 1 s for the velocity profile. A possible representant of a trajectory is illustrated in Figure 3-4.



Angle	φ_1	φ_2	φ_3	φ_4	φ_5
Degree	-10°	+9°	+48°	+12°	0°

Figure 3-4 Trajectory presentation containing path and velocity profile

3.1.4 Vehicle Control

As mentioned in previous chapters, vehicle control can be subdivided in a longitudinal and lateral part. The output of the CNN contains a velocity profile and a trajectory path with five steering angles. A simple cruise control assigns the velocity profile (output of CNN) to throttle and brake pedal position.

Steering control, on the other hand, is more difficult since not only a single but also multiple trajectory points are relevant to determine an optimal slope of the path for precise positioning. A first simple approach of a relatively direct steering control that was implemented adapted the steering wheel angle with the first trajectory angle only, multiplied by a constant compensation value. The value acts as an amplifier correcting the difference between the yaw and steering wheel angle. This works for simple scenarios where no precise driving is necessary. A more secure and versatile method is the implementation of the Stanley steering controller which is described mathematically by Formula (3-1) [43, p. 24].

$$\delta = \theta_e + atan\left(k \cdot \frac{e_{fa}(t)}{v_x(t)}\right) \tag{3-1}$$

The Stanley controller consists of two separate parts, starting with the heading difference θ_e which is calculated using the deviation of vehicle direction and slope of the nearest point P of the desired trajectory. The second term describes the lateral offset of the vehicle to the desired path. This is done by dividing the lateral error e_{fa} by v_x, the longitudinal velocity of the ego-vehicle. The constant k serves as a tuning factor that adjusts the stiffness of the vehicle regarding the optimal trajectory. The lateral error e_{fa}, as well as the directional θ_e error, are calculated with respect to the trajectory point that's nearest to the middle of the vehicle's front axle. The Stanley steering controller was developed by the winning team of Stanford University for the DARPA challenge. Since in the challenge speed was restricted to about 80 km/h, which exceeds the maximum speed within inner city traffic scenarios the Stanley controller should be considered suitable. [43, p. 24]

3.2 Discussion of the movement planning approach

In contrast to the widely discussed NVIDIA End-to-End approach a solution has been favored which allows modulation. This includes horizontal modifications by adding algorithms and solutions that perform best on their field, as well as a vertical extension to add additional environment representations the algorithm has to optimize its trajectory for.

This concept of splitting the overall task of autonomous driving concepts into several modules allows choosing the best method (conventional computer programs or deep learning algorithms) for each subproblem within the overall task. As mentioned above, the use of Neural Network has distinct advantages in some areas (e.g. generalization effect), but they are not favorable in some other computational tasks (many training data necessary, difficult validation). Especially, when there is a clear and straightforward describable logic in the focused tasks traditional computation has a clear advantage over Deep Learning. When generating better results with velocity determination of moving vehicles by applying a laser scanner exploiting the Doppler-effect in comparison to a dual camera module and a NN then the first solution should be selected and integrated. To enable the integration of variable components, a horizontal modular approach is used which comes with some various other decisive advantages compared to the End-to-End solution from NVIDIA as well.

Modular building blocks can be added: During development additional input information can be added to increase the performance of the algorithm or to adapt it to other tasks. During the first development stages, a basic vehicle behavior might be implemented for stable and secure vehicle movements. After solving this task, additional features as detecting and avoiding potholes will add additional comfort value to the solution.

The number of training examples can be reduced: As seen from various other automatized tasks a full End-to-End approach typically needs a broader set of samples for training to achieve the same performance as a subdivided system might requires. Since NNs can be considered as black box models, no clear explanations can be given for this behavior. Although it makes sense from a system's perspective since a more complicated task which cannot be divided into easier subordinate tasks enables less guidance and requires higher connectivity than a step-by-step approach.

Failure detection and error elimination: Even when adding more complex automated movement control for cars, failures like collisions or lane departing can occur. Since NN are considered as black boxes, their inner dependencies are difficult to understand. Finding the nodes which are responsible for the NN is still possible when reading the internal feature maps, although a failure can result from erroneous weights and biases on several connected nodes. Improving the NN to solve one particular erroneous situation manually is even more difficult since simply changing a parameter in a hidden layer which is connected to all nodes in the upcoming layers can result in newly generated errors [37]. A workaround might include extending the training set with data in which these problematic situations occur. This results in the necessity of additional training data which then can lead to a changed behavior that might decrease satisfaction concerning other previously mastered situations. NNs are sensitive to the data they are trained on, especially when trying to solve problems amongst minorly appear situations. A divide-and-conquer approach reduces complexity regarding one issue. When having different clearly separated steps which are defined manually by humans might result in a more easily error detection and improvement.

There is still a decisive downside to the modular buildup that should not be kept unconsidered.

Additional feature engineering: When designing a modular software solution, a crucial part is to define boundaries, which separates subsystems from each other [11, p. 6]. Each subsystem requires various outputs provided by preceding subsystems. When an element needs certain information to function on an optimal level which is not transmitted because of insufficient system engineering a loss in performance can be expected. In the case of movement planning, information about traffic light states might be required to master many scenarios. When no information is provided the system might underperform, resulting in unstable behavior. When using an End-to-End approach, all data are delivered if sensory inputs cover all necessary human senses. In a modular solution, these features must be handcrafted and therefore the risk of overlooking critical information might occur.

3.2.1 Implication on training data

When choosing a Neural Network for the presented task different issues have to be solved considering the data it will be trained on.

The Number of training data: An obstacle to overcome when using Neural Networks for any task is the immense number of training examples that are required to achieve good performance. Meaning that data quantity and quality are essential and have to be brought into focus. Neural Networks require a broad set of training data to find and acquire information about underlying logics [11, p. 90, 44, p. 240].

Training data quality: Quality wise training data can be seen as the ground information which is never be scrutinized by the network. False examples in the training-set, therefore, can have a negative impact on the performance of the network. It should be mentioned that Neural Networks tent to counterbalance randomized nonsystematic errors if the majority of data is correct. Since NN are similar to a black box the effect of these randomized errors to inner weight adaption and therefore overall output result is hard to predict in any situation. Consequently, it is recommended to have a clean dataset when it comes to safety-critical tasks. In a worst-case scenario training data contains systematic (biased error) in which case the network might find wrong or nonexistent patterns in data and therefore reacts in an unexpected manner [45].

During the beginning phase of the master thesis, a first idea was to gather recording data from driving sessions using a traffic simulation (SILAB). First data can be acquired fast and training can start quickly. Nevertheless, the amount of data that is required exceeds the available time for finishing the thesis massively. Only a single vehicle can be used for data generation. Any change in the maps must be adapted manually. Concluding an own more flexible simulation environment has been developed for data generation which will be introduced in the upcoming chapter.

4 Simulation environment and Ground Truth generation

One major part of this thesis was the development of a simulation environment. The primary task of this software is to create a large number of automatically labeled examples to train the CNN spoken about in the previous chapter.

This chapter is divided into three parts. In advance of the programming process, a listing of all requirements should give greater clearance about the goals of a suitable simulation environment (section 4.1). The solution is introduced, explaining the overall concepts by presenting all simulated items and main processes (section 4.2). To allow the user some guidance, the functions of the software are explained step by step (section 4.3). At the end of the chapter, equipped with basic knowledge about the software, some key features and solutions are described to allow a more profound understanding on complex parts of the simulations (section 4.4). Since it would massively exceed the page limitation of the thesis, no code is presented. Only essential concepts are described, even though, some of them require advanced mathematical or computational knowledge for implementation.

4.1 Requirements

In advance to programming a simulation environment, requirements are necessary to fit the needs of the movement planning algorithm and applying users.

Generating large sets of training data: As been mentioned before, CNNs require a significant amount of training data to perform reasonably well. During the simulation process, these data should be generated with a minimized working time for the user. Variation and multiple application of generated scenarios play a significant role. Additional time dependency in traffic environment can be exploited to save various training data monitoring even few vehicles [11, p. 90, 44, p. 240].

Enabling high data quality: Since NN learn by comparing their prediction with the labeled training data, failures and insufficiencies are trained to replicate as well. The term Ground Truth comes from the fact that a network does not question the presented training data, but instead regards them as correct information. Thus, reducing errors including collision detection and high accuracy calculations (tolerance of under 1 cm), etc. are essential requirements for the software. [45]

Easy to extend: During midterm of this project, written C# code has been translated into the beginner friendly and more Python like VB.net language. Especially, allowing mechanical engineers without any advanced programming skills to extend the software for their projects.

Easy to use: Finally, a sufficient GUI improves positive user experience allowing fast and comfortable working and reduction of perceived subjective complexity. A GUI concept manly includes a visual presentation, feedback and interaction with simulation setups. Furthermore, the user should be supported with various background functions to limit the complexity of the software. For instance, when defining polygons, the right order of points is requested to enable expected results. Solutions like the Graham Scan algorithm can reorder these points to support the user.

4.2 Simulation framework

The developed simulation framework software was written in C# and later converted to Visual Basic.net, as mentioned. Mainly because VB.net reads familiar to engineers at FTM joining the project use Python and TensorFlow to program their Networks. Furthermore, Visual Basic.net benefits from Microsoft's extensive .net Framework which allows building GUIs easily and fast. The self-written simulation software contains about 40,000 lines of code and 17 GUI forms. To add additional GUI tab functionality to vanilla windows forms solution the NuGet element "WeifenLuo WinForms UI Docking" has been included as the only external package.

In the next chapters basic concepts of the simulation environment will be described. Starting with an overview of implemented items will depict the potential and restrictions of the software (section 4.2.1).

Decreasing complexity and giving guidance to the user the overall Data generation process is basing on three underlying elements. Superscenario, Scenario and a Subscenario are generated within the simulation framework (Chapter 4.2.2).

4.2.1 Simulated items

As mentioned, in earlier chapters, a modular approach to autonomous driving, unlike an End-to-End concept requires additional feature engineering. In this case, elements that might have an impact on AI's behavior must be known apriori. Therefore, the selection of these items must be implemented into the development of the simulation framework avoiding limitations in the performance of the back-end AI solution. The succeeding list of elements has been worked out to accomplish this result. All listed items are implemented in the simulation environment.

Road markings: Not to be confused with road lines, road markings give the vehicle guidance in the lateral direction and limits the legal driving space. Road markings include basic lane markings (dashed and full lines, cubic and hermit splines) but also all sorts of markings (Arrows and pedestrian crossings). To allow extending marks list, the user can add his own elements into a library.

Moving vehicles: The simulation of traffic is the primary focus of the simulation environment. Vehicles follow the rules of road traffic act by reacting to user-defined logics like yielding or dynamic traffic light states, as well as other vehicles and pedestrians nearby. To reduce complexity vehicles are restricted to move on predefined tracks that include a velocity profile which is targeted when no reaction to the environment occurs. Basically, vehicles are differentiated between ego- (focused) and third-party-vehicles. Training data is generated when a focused ego-vehicle is located on a predefined area and is moved by a certain time step. To enhance the collection of training data each vehicle considered an ego vehicle, once in a time step. Vehicle states and surroundings are then combined to a single data point, to train the NN on. Vehicles are spawned randomly on the starting point (sources) of a route and despawn after finishing their trip (sink).

Moving pedestrians: As pedestrians are the weakest participants within the road traffic they require particular focus by a self-driving car. Differentiating from vehicles, they are described by another movement model, since fewer restrictions can be made concerning their heading velocity changes. Their behavior is randomized so that vehicles have to react to them in order to prevent collisions. Just as vehicles, pedestrians spawn randomly within the focused area of the map but with predefined waypoints and velocity profiles.

Traffic lights and signs and logics: Logics help to regulate the traffic. Each logic is characterized by a switch point laying on a vehicle route and a Boolean logic element which determines whether the switch point is active or not. When it is active the car will decelerate and stop at its location on the route. Logic elements include time-related switching (traffic lights) or track-query switching (stop or yield signs).

Obstacles and environment: An environment should not increase performance but adds noise to the images. Surrounding building blocks, sewer coverings or bushes and plants not serve a purpose considering prediction, or trajectory planning. Contrarily, it may interfere system’s capabilities. Nevertheless, they need to be added to the data otherwise data mismatch between training and runtime data may occur, which might destabilize the system in validation.

With the list of simulated elements above, several relevant items have been neglected to ensure manageability in the thesis. These include bicycles and other two-wheeler, Trucks with trailers, busses, as well as reaction to bus stops as well as ambulances or other priority vehicles. As mentioned, both the simulation framework as well as the network are capable of dealing with implementing further items in future projects.

4.2.2 Simulation process

In each simulation process, different steps have been undertaken to initialize and finish training data for Deep Learning.

Superscenario: Starting with the underlying map, the Superscenario describes the overall architecture of the traffic element, containing road crossings, roundabouts using traffic lights, signs or basic yielding. These Superscenarios will be designed using basic geometric objects like points, lines or polygons, and values including lane width, angles, etc.). It also contains information about possible vehicle movements (tracks) and describes only a single traffic scenery to be easily changeable (T-intersection, roundabout with 4 connected streets etc.). To simplify organization processes, a second level has been added which is referred to as versions. It is often necessary to have smaller changes to a map, that is best represented on the super scenario level. For instance, a T-intersection might have an additional turning lane, or vehicles parking on parking bays. These variations are best represented in a Superscenario but not necessarily represent an independent new one.

Scenario: Training a NN requires a large number of training data with diversity playing a pivotal role [44, p. 240]. It is not recommended when trying to achieve a good generalization effect to train the Network with, varying data. Deploying only a small number of maps for data generation may lead to redundant traffic situations. On the other hand, map designing is very time-consuming, even with support tools built into the software. A diversification process was added to change different features and generate combinations of variation groups. During a feasibility check, the user can find patterns and delete impropriate combinations. Thus, single

Superscenarios can result in hundreds of different feasible Scenarios that are then utilized for Subscenario generation.

Subscenarios: After defining all map elements the time-dependent traffic simulation can begin to generate training data eventually. A Subscenario can be considered a traffic situation with various third-party vehicle and pedestrians surrounding the focused ego vehicle. Subscenarios are created during a simulation session using the Scenario map architecture. A Subscenario is then saved containing all relevant information for further use, including environment representations and Ground Truth.

4.3 Stepwise instruction

In this chapter, a step by step explanation is provided, on how to generate a simulation and extract Ground Truth from it to train the network. The focus lays on the core process, minor details and comfort functions are not described in order to keep the subject manageable. Essential features are mentioned in this chapter but are taken up and explained in the upcoming sections.

Before beginning to describe the features of the software, some preparations have to be done to the PC to ensure continuous impaired working. The software, as well as the upcoming subsections, are sequentially dealing with designing the Superscenario (map), Scenarios (variations of maps) and Subscenarios (simulation).

4.3.1 Preparation and GUI overview

Starting with some preparations, European PC setup have to change the standard numeric format. This can be achieved by adjusting numeric formats under *Control Panel*, *Region*, *Format*, *advanced settings* (in Windows 10) or *Control Panel*, *Language and Region*, *Additional settings* (in Windows 7). In the shown form (*Customize Format*) the properties have to be changed to those denoted in Table 4-1.

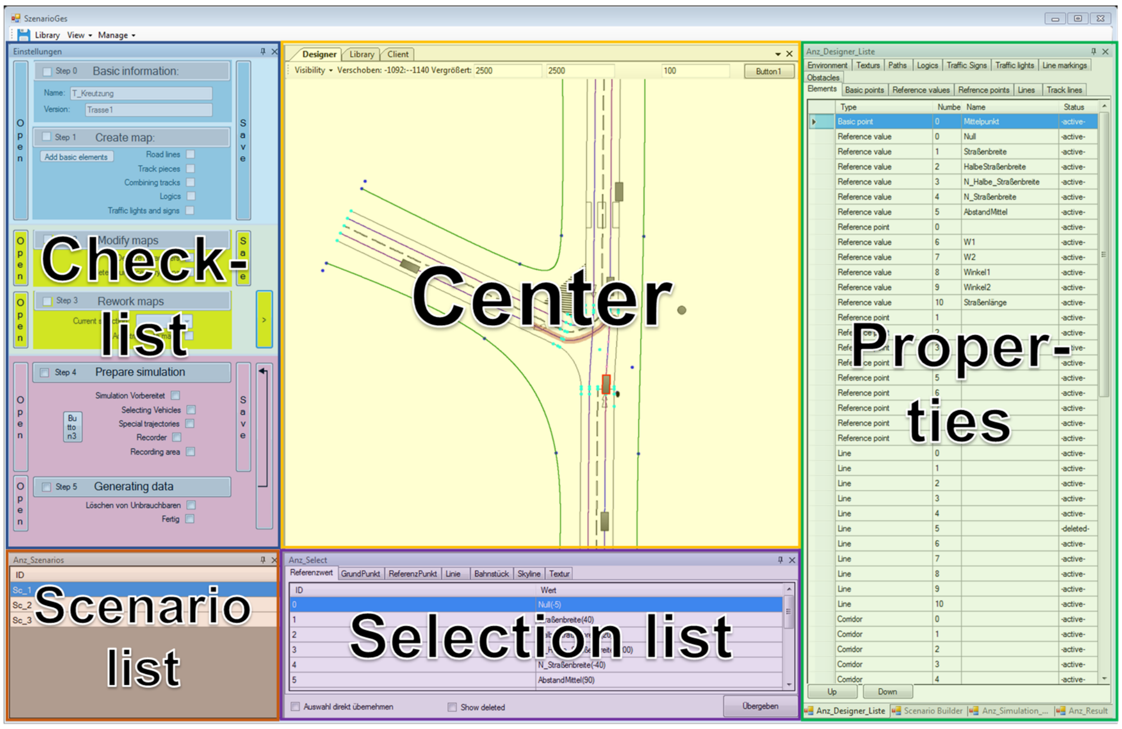
Table 4-1: Required Windows property settings

Property	Selection
Decimal symbol	. (Point)
Digit grouping symbol	, (Comma)
Negative sign symbol	- (Minus)
List separator	, (Comma)

After setting up Windows, it is recommended to create a working directory, preferentially on a network drive to allow decentral computing for simulation purposes (chapter 4.3.6).

A short overview of the software should be given in order to have an understanding of where to find certain GUI elements. In this documentation, only more complex interactions are stated. Various elements that are easy to understand are not covert within the master thesis.

Starting from the left side, a checklist is provided to guide the user, showing current working steps and document progression. In the center, the main designer and visualization element is located. The designer allows zooming and scrolling of maps and selecting points. The selection list in the lower middle shows all available elements. On the right side depending on the simulation step, an object list (Section 4.3.2), variation properties (Section 4.3.3), simulation preparation (Section 4.3.4), or simulation results (Section 4.3.5) is shown. To create or change properties of Objects it is often relevant to pick already existing elements (e.g. for generating a line, selecting two points is necessary). The user can either click on a certain point on the designer (center) or selecting and applying element within the selection list for more precision. Apart from number values that can only be found in the selection list, it is up to users' preferences which way to choose for selecting elements. When multiple maps are available, a list appears in the lower right to enable fast switching. In Figure 4-1 the full UI is presented.



Name: T_Kreutzung
Version: Trasse1

Verschoben: -1092:--1140 Vergrößert: 2500 2500 100

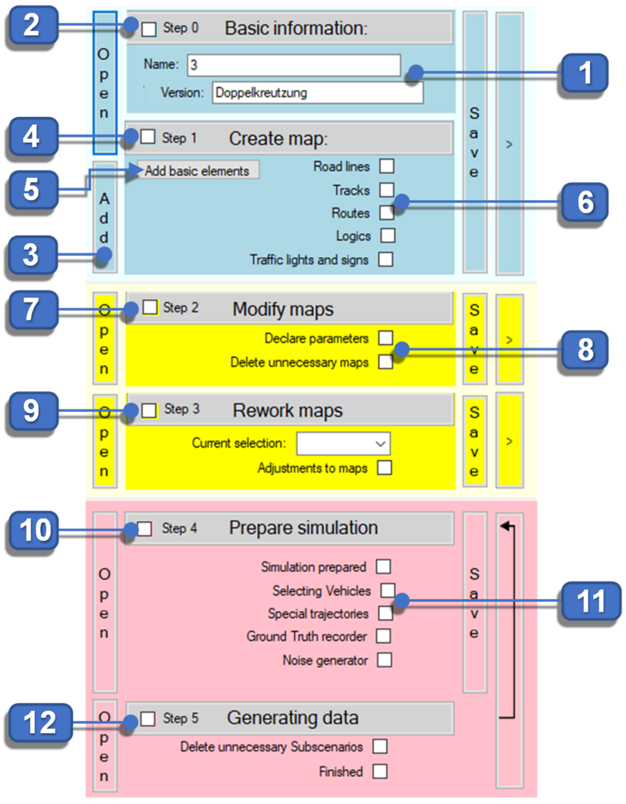
Type	Numbe	Name	Status
Basic point	0	Mittelpunkt	-active-
Reference value	0	Null	-active-
Reference value	1	Straßenbreite	-active-
Reference value	2	HalbeStraßenbreite	-active-
Reference value	3	N_Halbe_Straßenbreite	-active-
Reference value	4	N_Straßenbreite	-active-
Reference value	5	AbstandMittel	-active-
Reference point	0		-active-
Reference value	6	W1	-active-
Reference value	7	W2	-active-
Reference value	8	Winkel1	-active-
Reference value	9	Winkel2	-active-
Reference value	10	Straßenlänge	-active-
Reference point	1		-active-
Reference point	2		-active-
Reference point	3		-active-
Reference point	[illegible]		-active-
Reference point	5		-active-
Reference point	6		-active-
Reference point	[illegible]		-active-
Reference point	[illegible]		-active-
Reference point	[illegible]		-active-
Line	0		-active-
Line	1		-active-
Line	2		-active-
Line	3		-active-
Line	4		-active-
Line	5		-deleted-
Line	6		-active-
Line	7		-active-
Line	8		-active-
Line	9		-active-
Line	10		-active-
Corridor	0		-active-
Corridor	1		-active-
Corridor	2		-active-
Corridor	3		-active-
Corridor	4		-active-

ID
Sc_1
Sc_2
Sc_3

ID	Wert
0	Null(-5)
1	[illegible]raßenbreite(40)
2	[illegible]
3	[illegible]
4	N_Straßenbreite(-40)
5	AbstandMittel(90)

Figure 4-1 Overview of the GUI of the simulation environment

The first relevant GUI element the user is interacting with when starting a project is the check or guidance list on the right, illustrated in Figure 4-2. As previously discussed in chapter 4.2.2, the simulation process is subdivided into Superscenario, Scenario and Subscenario creation. These steps are represented in the guidance list as well, using color coding (blue = Superscenario, yellow = Scenario, and red= Subscenario). All steps can be loaded, saved and continued (">" Button) within the UI. When first starting a project, it is advised to name it in advance (1). Superscenarios are identified by their name and version. In addition to loading a map (2), the user can add existing maps (templates) into its current Superscenario. For instance, if three-road interception should be designed, road templates can be loaded into the Superscenario, and attached later. All variables and elements will be copied to allow variation afterward (3). During usage, it turned out that the name best describes the overall Type (Straight, S_Curve, T_Junction, 4_Crossing, Roundabout etc). The version, on the other hand, was applied to give some more information about the specification (Traffic_light, Traffic_Signs, Pedestrian_Crossing, Refuge_Island, etc.). Next, the design process for the named Superscenario begins (4). When starting from scratch often used elements can be comfortably added via button click (5). Starting with road lines, vehicle tracks and routes, logics, traffic lights and signs, Environment and obstacles all relevant items as described in chapter 4.2.1 will be drawn onto a map (6). For reapplied elements like Road markings, buildings or obstacles a library has been implemented that can be extended by project relevant items. A more detailed look at the library and their elements will be given in chapter 4.3.2. Elements of the included library are introduced when their elements become relevant during the map design process. The full library can be found on the equally named tool-tip button on the upper left side of the main window. After the designing phase, the variation process starts to duplicate and manipulate Scenarios in order to increase diversity by changing street courses (7). This is accomplished by the steps of parameter declaration and variation as well as deleting inappropriate maps (8). If additional rework is required, it should be held to a minimum (9). At this stage design process has finished and necessary settings for the upcoming traffic simulation should be adjusted (10). This includes defining time span and interval, training data recorder and noise overlapping (11). Simulation process can be executed with UI support (12) or using the Sever-Client solution (Chapter 4.3.6). It is recommended to check the results of at least a single simulation with the integrated UI tool.



Step 0 Basic information:

Name: 3

Version: Doppelkreutzung

Step 1 Create map:

Add basic elements

Road lines
Tracks
Routes
Logics
Traffic lights and signs

Step 2 Modify maps

Declare parameters
Delete unnecessary maps

Step 3 Rework maps

Current selection:
Adjustments to maps

Step 4 Prepare simulation

Simulation prepared
Selecting Vehicles
Special trajectories
Ground Truth recorder
Noise generator

Step 5 Generating data

Delete unnecessary Subscenarios
Finished

Figure 4-2: User guidance list GUI

In the subsequent chapters, the upper overview of the working process will be explained in greater detail.

4.3.2 Superscenario design by map generation

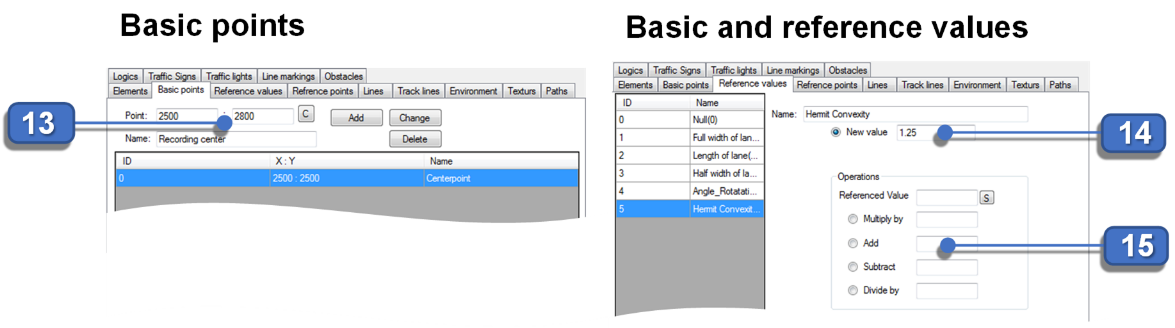
As mentioned before, the basis of a simulation is a designed map. It is built up from geometric points and values combined to lines and more complex polygons. This helps to generate easily adjustable maps. For instance, changing a single value like an angle of a street intersection results in high visible diversification using simple means. This comes with the costs of a more complex designing process. This part of the software called Map designer should enable and support the user to generate mentioned maps. It is located at the properties area on the right side of the UI. When starting a new Superscenario, one first faces the element list. If the map is still empty, elements are not registered and the list is therefore unimportant for the user at this stage. Therefore, its explanation is covered at the end of this section. Instead, the first focus is set to basic points and reference values each other structure evolves from.

Basic points and values

A basic point defines the middle of the road network and therefore is the starting point of map design. It is advised to only generate a single basic point for any Superscenario, if just a single traffic situation is involved. Furthermore, it is recommended to consider it as the center point. When adding basic elements to the Superscenario a basic point is already set into the middle of the map (4).

In the next subsection a distinction is made between basic and referenced elements. Basic means that they are newly generated and freely defined by the user (13). A Basic point, therefore, does not include any constraints considering its location, while reference points refer to a set of points including a manipulation operation.

Basic values describe, similar to basic points, elements that are not connected and numeric user inputs (14). Referenced values require a reference in combination with a mathematical operator (15). An overview regarding all options is given in Figure 4-3.



Point: 2500 : 2800

Name: Recording center

ID	X : Y	Name
0	2500 : 2500	Centerpoint

ID	Name
0	Null(0)
1	Full width of lan...
2	Length of lane(...
3	Half width of la...
4	Angle_Rotatati...
5	Hermit Convexit...

Name: Hermit Convexity

[✓ New value] 1.25

Referenced Value

Multiply by

Add

Subtract

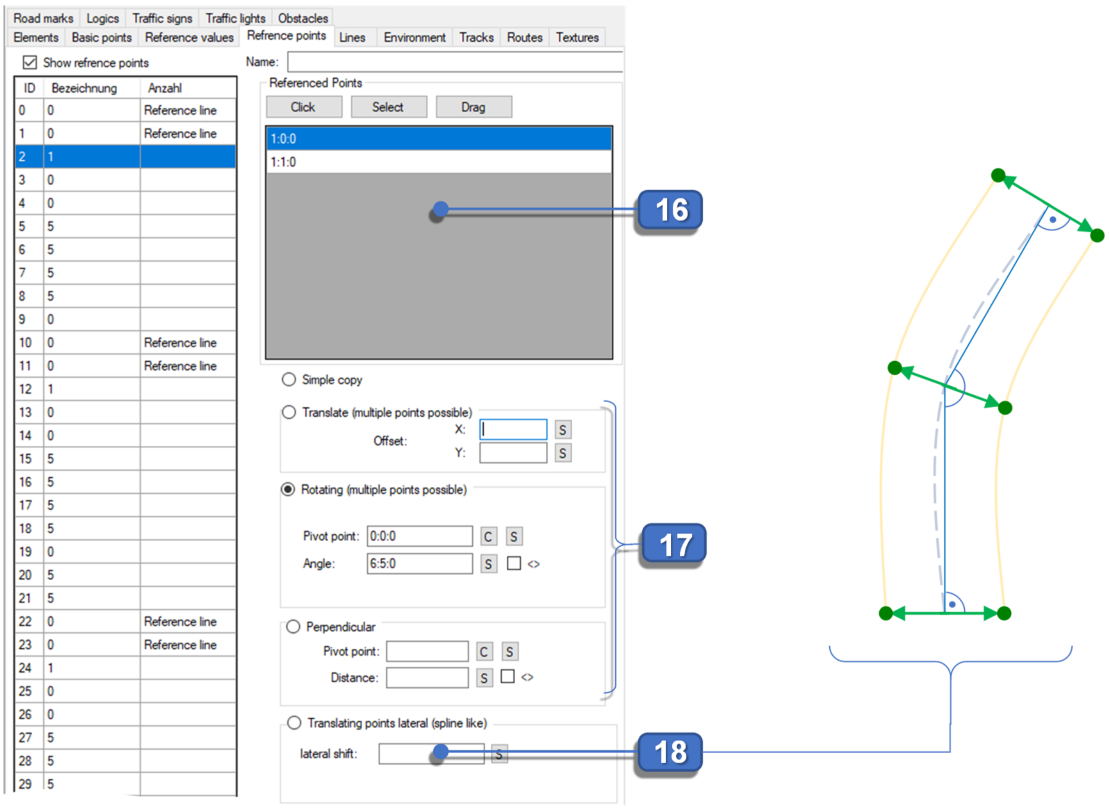
Divide by

Figure 4-3: GUIs of basic points and values

Reference points

Similar to the discrimination of value types, reference points can be considered a manipulated clone of a basic or other reference points, using more complex geometric operations. These related points have to be selected by clicking on them in the designer UI or selecting them in the selection list (16). Four possible mathematical operations are available. Translation, rotation and perpendicular rotation (17) may not need any further explanation. Lateral translation of points moves points towards the outside of a spline curve (18). This allows faster designing processes considering curvy street sections. The GUI presenting reference points settings is illustrated in Figure 4-4.

Reference points



☑ Show refrence points

ID	Bezeichnung	Anzahl
0	0	Reference line
1	0	Reference line
2	1	
3	0	
4	0	
5	5	
6	5	
7	5	
8	5	
9	0	
10	0	Reference line
11	0	Reference line
12	1	
13	0	
14	0	
15	5	
16	5	
17	5	
18	5	
19	0	
20	5	
21	5	
22	0	Reference line
23	0	Reference line
24	1	
25	0	
26	0	
27	5	
28	5	
29	5	

Name:

1:0:0
1:1:0

○ Simple copy
○ Translate (multiple points possible)
Offset: X: Y:
◉ Rotating (multiple points possible)
Pivot point: 0:0:0
Angle: 6:5:0
○ Perpendicular
Pivot point:
Distance:
○ Translating points lateral (spline like)
lateral shift:

Figure 4-4: GUI regarding reference point options

Lines and street environment

After defining all necessary point elements, lines can be drawn onto the map. To start with, it must be explained that several elements exist which appear as a line at first but have different functions. A basic line simply helps the vehicle to identify drivable areas and distinguish the road from environment surroundings. Environment lines help the user by the positioning and alignment of surrounding constructions like buildings, bushes or pillars. Track-lines are used for defining trajectories for vehicles, including location and velocity guidance.

In this section two representants are described in more detail, starting with basic visual lines and introducing environment lines later on. Three different types of graphs exist to accomplish building more complex map architectures. The most complex, is a (cubic) spline which acquires at least three support points and appears as a curved interpolated line connecting them (19). An often-used element to connect intercepting lines (side lines at a street crossing) and design a curved junction is a hermit spline (20). It is defined by two end-points (supports 20.1) each combined with a tangential aligned point to determine the slope of the curve (20.2). A convexity value characterizes the round shape of a hermit spline. A large value results in an edgy spline, with the curved transition moves away from points to the calculated interception of both support lines. A zero value results in the opposite shape, which approximates to a straight line starting and ending at support points. Last, the straight line by itself requires two points a user has to select to connect them linearly (21). Since lines are considered as objects they can have different properties, including a line-type (full line or dashed, 23) or section-wise visibility (22).

Each line consists of 100 subpoints (apart from splines that include 100 points per section) which are used for rendering the curvy appearance. Every subpoint defines the properties of the attached line element. This allows the user to modify the graph's appearance precisely. Each point is calculated whenever a change is made to the line. Therefore, it moves accordingly to the host line's location and orientation. When designing maps this can be exploited by connecting other elements to these point for the generation of a smooth transition between lines. if a point of a line is used for defining other lines or other element's position this point is further referred to as active points, the line is referred to as host line. They are made distinguishable from other points or lines by a highlighting cyan circle surrounding them. Later on, a tool is presented which allows moving these points on the host line to correct elements or decrease diversity amongst Scenarios.

Environment lines are applied to align and draw construction and objects to a map. Environment lines and (road) lines are stacked together in this subsection since they are generated the same way. Both base on the same graph types, splines, hermits and straight lines. They only differ in the fact that environment lines are direction-dependent. Constructions always appear on the right side of environment lines (24). Therefore, an additional function has been added to quickly check the direction by adding some environment construction to the scenery and change direction when needed (25). To show and add additional buildings the library is available via a button click. The user is then redirected to the construction section.

In the middle of the library UI, a checkered picture box is present which shows the polygonal shape off the selected building (25). New points can be added with a click on the grid which is then added to the list on the right (26). Apart from the contour, some other information is relevant to draw the object to a map, starting with the reference point that appears as a red curricle on the picture box when selected. This point is relevant for docking and aligning a construction onto an environment line (24, 27). Secondly, a distance has been entered in which no other construction object can spawn on an environment line. It restricts overlapping of buildings (28). The environment line is randomly occupied with objects. This generates higher diversity in training data since the scenery is changing in every data point. All interactive user elements are illustrated in Figure 4-5.

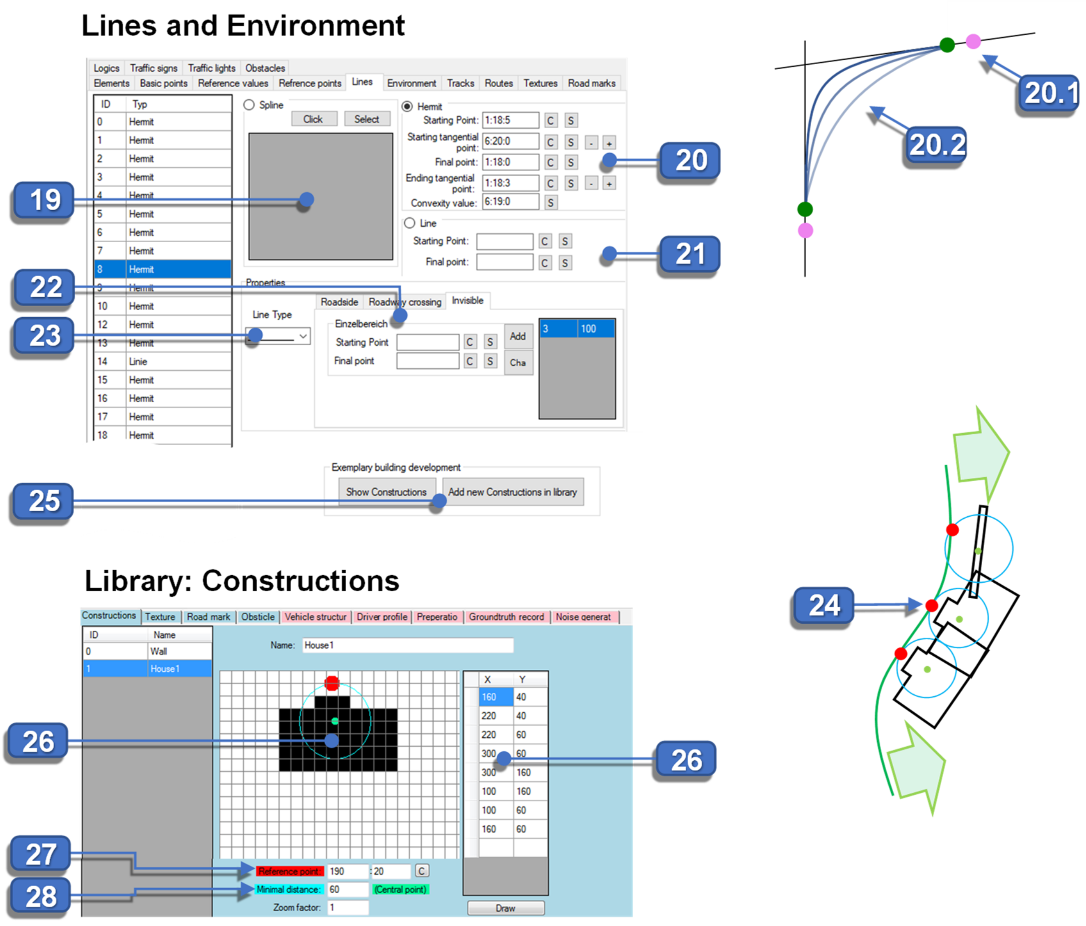


Figure 4-5: GUI covering lines and environment

Tracks and routes

Vehicle movement simulation requires guidance for vehicles. For this reason, tracks and routes have been added to the tool compilation. While tracks characterize corridors using simple line elements known from the previous subsection a route describes the full routes a vehicle takes from start to end of the map. Routes are generated by adding connected tracks to them.

Basically, the form of tracks is equal to other graphs and therefore are generated by the same three types and the known procedure. They are direction-orientated as well, which implies that vehicle move from the starting point towards the ending points. This does not necessarily mean that they have to entrance at the first point or leave at the last point as will be shown later. The direction can be shown by drawing arrows heading in the leading direction of the tracks (29). A switch button reverses the direction of the line if correction is needed.

Apart from lateral a vehicle also requires longitudinal guidance to reach its destination. When a vehicle is following a preceding one, speed is limited by the car in the front. In a free driving situation, other rules have to be followed including speed regulations or comfort resulted limited lateral forces while passing a curvy road. Decelerating before entering a traffic light or construction side scenario also have to be taken into consideration when creating a simulation environment. Hence, the user is able to set the desired free driving speed by creating a velocity profile within a graphical interface (30). Left clicking adds a new constraint (support point) to the profile, right-clicking deletes it. With holding the left mouse button, one can move a support point. A 1d-cubic spline interpolation then increases the resolution and generates the full velocity profile. When saving the profile, the track is colored according to the desired velocity.

As already mentioned, routes map out the full path a vehicle takes from spawn to exit. They can be extended by selecting tracks in the order form a route through the map (31). An algorithm which supports the user to determine optimal connections has been added. It simply finds the points that have the smallest distance between them (32). A vehicle always entrances a route with the starting point and despawns at the final point respectively considering a tolerance (usually 30 points). This automatic route calculation system is necessary when points are moved to change street courses in Scenario variation process described in section 4.3.3. For visual reasons, loose tracks can be automatically cut to remain their drivable sections (33). All options regarding tracks and routes are illustrated in Figure 4-6.

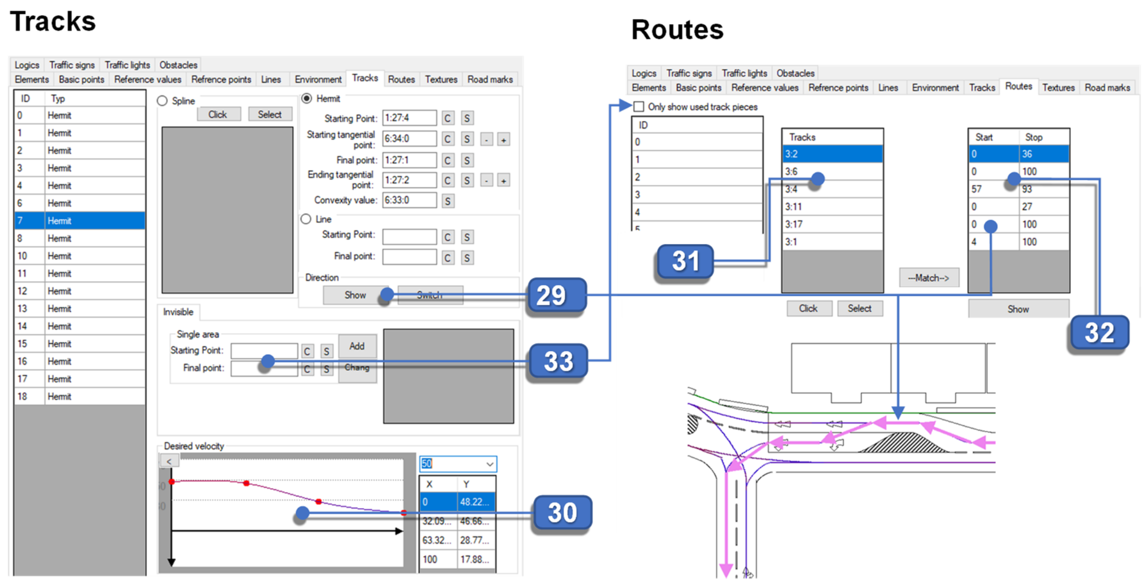


Figure 4-6: GUI including track and route settings

Surface textures and road markings

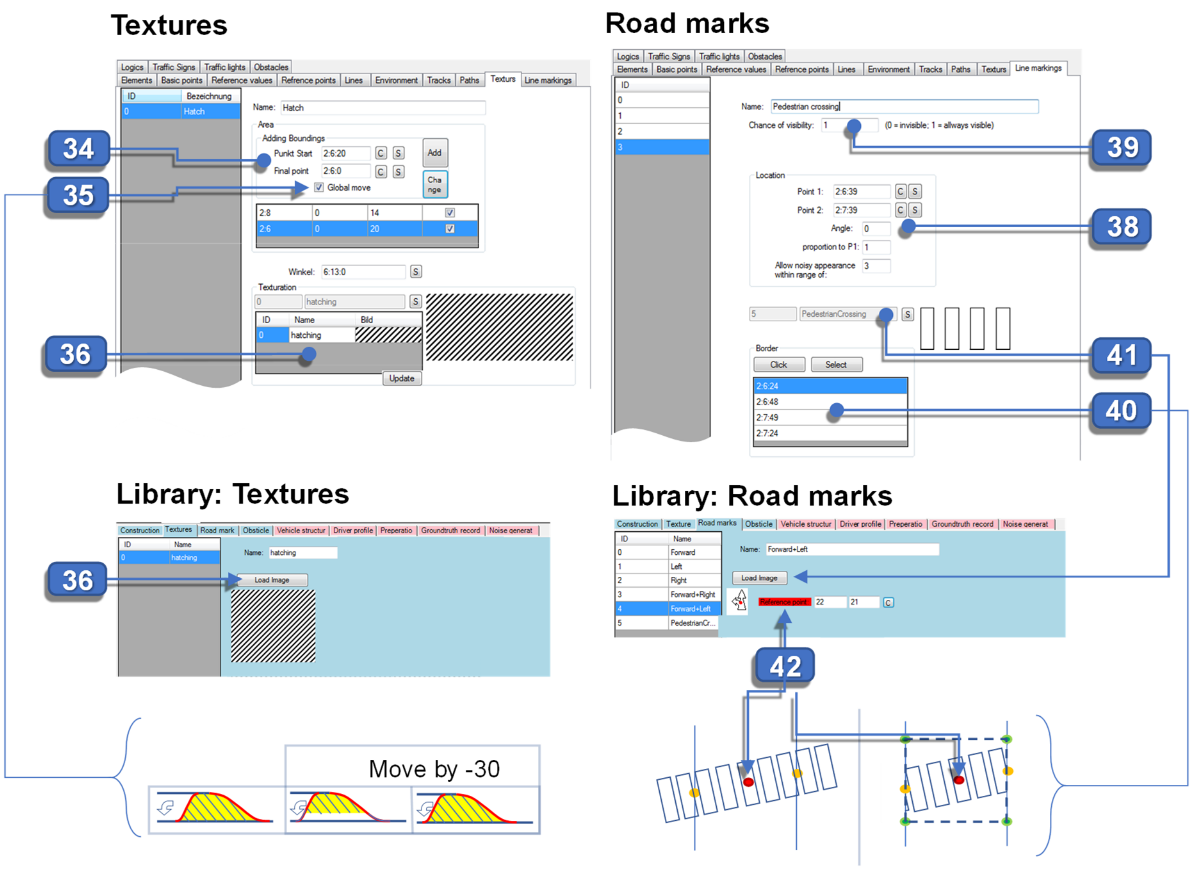
Often elements or patterns are drawn on the street surface which have an additional meaning to the driver. These are represented in this software as textures and road marks.

Textures are defined by a surface painting that is bound by a polygon. This polygon is built with a set of points from different already drawn lines or tracks, surrounding the new texture. This is done by selecting a start and end point of an element (mostly lines) which contributes to the border (34). Later on, a tool will be described which allows moving active points on lines. In this way, for example, turning lane restrictions can be moved along a line. The striped surface is an example of a texture that is bounded by two aligned hermit splines left and right. When moving all lines by 3 m, this would result in moving the start point of the bounding area of the texture as well leaving a blank slice below the striped surface. To prevent this, the user can uncheck the global move checkbox for the selected bounding. The starting and ending points of the texture's bounding will not be moved on a line if this property is set (35).

To accurately define the bounding of a textured area often sub-points from multiple lines are required to set a polygon. Polygons are sensitive to the order of the defining points. Four points, for instance, might define, a square, an eight-shape like polygon, depending on the order of the points. Hence, an algorithm should support the user to bring the polygon in the shape of a mostly desired convex hull. The Graham-Scan algorithm has been implemented to provide a solution for this issue [46].Since textures are used repeatedly in many projects, an entry in the library has been added. Here the user can import a texture bitmap (black and white only preferred, 36).

In contrast to textures, marks are images that are painted on a specific selected spot onto the street. (Pedestrian crossing and guiding arrows). Therefore, the location process is designed differently. Two referenced points and a proportion value is required to define the position. An angle sets the direction of the marking (38). Additionally, an appearance value is applied for decreasing diversity within a simulation (39). It indicates the chance that the markings are visible at any image. The decision for the appearance will be recalculated for any training example that will be created during the simulation process. For further increasing diversity, selected location points can be randomly reassigned while simulations. The user can modify the range of this automatic adjustment. A value of 3 means that both location points can be randomly variated by three positions on the host line. The calculation is training data specific equal to the chance of appearance. During the development of the software, it was found useful to add a bounding for road marks as well (40). When adding more extended directional elements like a pedestrian crossing a bounding help to define its expansion.

Markings can be saved in the library analogically to textures by importing an image bitmap (41). Besides, markings require a reference point for aligning purposes, similar to constructions as shown before (42). All mentioned settings are covered in Figure 4-7.


Textures
Road marks
Library: Textures
Library: Road marks
34
35
36
39
38
41
40
36
42
Move by -30
Logics | Traffic Signs | Traffic lights | Obstacles
Elements | Basic points | Reference values | Refrence points | Lines | Environment | Tracks | Paths | Texturs | Line markings
Construction | Textures | Road mark | Obsticle | Vehicle structur | Driver profile | Preperatio | Groundtruth record | Noise generat
Add
Change
Update
Click
Select
Load Image

Textures:

ID	Bezeichnung
0	Hatch

Name: Hatch
Area – Adding Boundings
Punkt Start: 2:6:20
Final point: 2:6:0
[✓] Global move

2:8	0	14	✓
2:6	0	20	✓

Winkel: 6:13:0
Texturation: 0 hatching

ID	Name	Bild
0	hatching	▨

Road marks:

ID
0
1
2
3

Name: Pedestrian crossing
Chance of visibility: 1 (0 = invisible; 1 = allways visible)
Location
Point 1: 2:6:39
Point 2: 2:7:39
Angle: 0
proportion to P1: 1
Allow noisy appearance within range of: 3
5 PedestrianCrossing
Border: 2:6:24, 2:6:48, 2:7:49, 2:7:24

Library: Textures

ID	Name
0	hatching

Name: hatching

Library: Road marks

ID	Name
0	Forward
1	Left
2	Right
3	Forward+Right
4	Forward+Left
5	PedestrianCr...

Name: Forward+Left
Reference point: 22 21

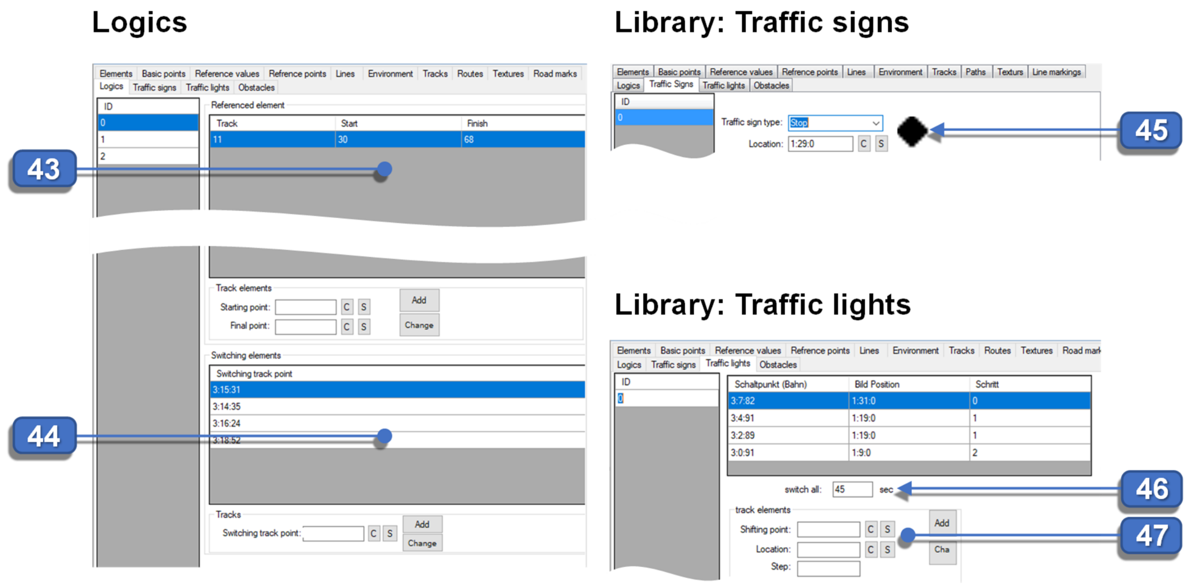
Figure 4-7: GUI elements focusing on surface textures and road markings

Logics and traffic signals

Track and routes provide vehicles with major road guidance. Many traffic situations still require additional interactions with other vehicles and surroundings. This can be programmed by computational elements, containing logics, traffic lights, and signs.

First of all, the basic concepts of logic blocks have to be understood. They are placed on tracks and contain two elements, detection and shifting points. First check if a vehicle is currently occupying its track-point, or more precisely is located on its predefined subtrack during simulation (43). If it detects a vehicle shifting is triggered, usually positioned on a different track (44). An approaching vehicle will decelerate, stop and wait until the track is free again. This allows, for instance, controlling traffic flow and vehicle priority on narrowed streets due to roadworks, or basic left yields to right situations. Traffic signs are simple pixel maps that are drawn to the referred traffic lights and signs as an own NN input. Their implication for the traffic has to be implemented by logics (45).

Traffic lights are characterized by timer related triggering of shifting points. Therefore, a shifting time must be defined (46), as well as the shifting point and location of the traffic light symbol for the environment representation (47). Defining states instead of single points allows switching multiple lanes in parallel. The shifting points are inverted in order to allow only one state including their lanes to drive. The GUI elements regarding logics as well as traffic lights and signs are illustrated in Figure 4-8.



ID
0
1
2

Track	Start	Finish
11	30	68

Starting point:
Final point:

Switching track point
3:15:31
3:14:35
3:16:24
3:18:52

Switching track point:

ID
0

Traffic sign type: Stop
Location: 1:29:0

ID
0

Schaltpunkt (Bahn)	Bild Position	Schritt
3:7:82	1:31:0	0
3:4:91	1:19:0	1
3:2:89	1:19:0	1
3:0:91	1:9:0	2

switch all: 45 sec
Shifting point:
Location:
Step:

Figure 4-8: Logics, traffic lights, and signs settings

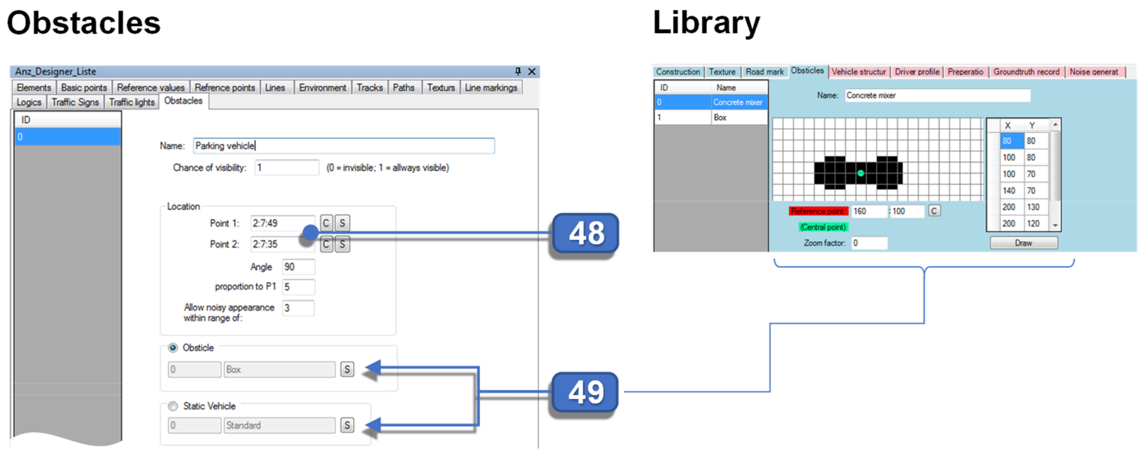
Obstacles

Obstacles are static objects that have no further purpose than to add noise and variation to maps. Positioning and orientating procedure is equal to road marks. Using two points and an angle, define the location and direction. Settings for the chance of appearance and allowed point variation increase diversity, as explained previously (48).

Two types of obstacles are available in the simulation environment and can be added to maps. First are static obstacles without any shape restrictions or further purpose. They are created and saved in the library. The designing process of a static object was held similar to the constructions apart from not requiring a minimal distance (49).

The second type of obstacles can be considered as fake still standing vehicles characterized by a vehicle's structural appearance. They can be selected, generated and saved in the library as well. Since vehicle structure is more relevant to traffic simulation, this library entry will be explained in greater detail in section 4.3.4. At this point, it's only relevant that the user can add vehicle structures and characterize them with a width and length which are then drawn as static obstacles. Standing vehicles are shown as regular vehicles but receive no velocity shading as discussed in section 3.1.1.

The appearance of obstacles in the training data depends on their type. Static obstacles are drawn on the obstacle map while standing vehicles are added to the vehicle map. The GUI for obstacle integration is illustrated in Figure 4-9.



ID: 0

Name: Parking vehicle

Chance of visibility: 1 (0 = invisible; 1 = allways visible)

Location

Point 1: 2:7:49

Point 2: 2:7:35

Angle 90

proportion to P1 5

Allow noisy appearance within range of: 3

[✓ Obsticle] 0 Box

Static Vehicle 0 Standard

ID	Name
0	Concrete mixer
1	Box

Name: Concrete mixer

X	Y
80	80
100	80
100	70
140	70
200	130
200	120

Reference point: 160 : 100

(Central point)

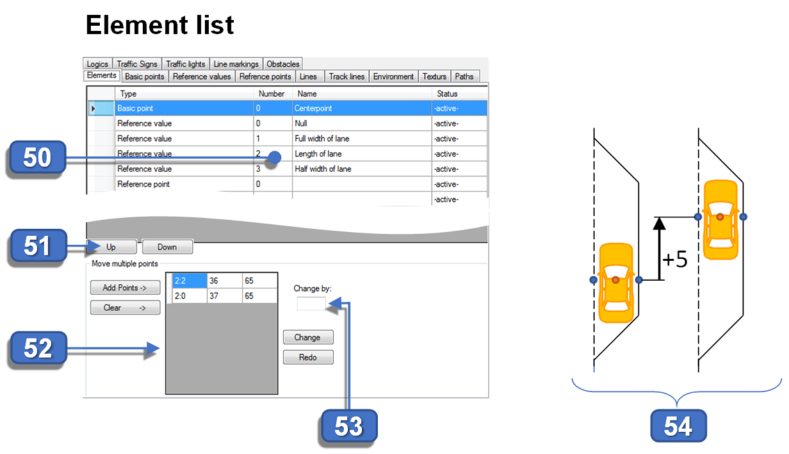
Zoom factor: 0

Figure 4-9: GUI for designing and implementing obstacles

Element list

After developing an overview of the deployed objects and adding several elements to the map, the element-list in the first tab becomes relevant, as it is illustrated in Figure 4-10 (50). All components are calculated and drawn sequentially, starting with the first added element (usually a basic point). When generating a new element, it will be added to the list as last row. Occasionally, it is necessary to add an object to the list that is computed before another already existing object has been generated. For instance, a hermit spline should have a separate convexity value. This value has been added as the last step. Therefore, a function has been added to manipulate the list manually and change the computational order of selected items (51).

After finishing a Superscenario it might be necessary to relocate single or groups of elements. Therefore, a function is provided which allows moving elements which points are located on a selected area of a host line (52). A distance (in points) then defense the number of points an element is moved on the host line. For instance, the two locating points of a parking vehicle sit on different host lines ($P1=3.0.15; P2 = 3.1.19$). When the areas of the host line/s are selected, and the movement operator is applied with a distance of 5 (53) the vehicle is moved towards the direction of the host line ($P1=3.0.20; P2=3.1.24$). This effect is visualized in Figure 4-10 (right, 54).



	Type	Number	Name	Status
▸	Basic point	0	Centerpoint	-active-
	Reference value	0	Null	-active-
	Reference value	1	Full width of lane	-active-
	Reference value	2	Length of lane	-active-
	Reference value	3	Half width of lane	-active-
	Reference point	0		-active-
				-active-

2:2	36	65
2:0	37	65

Figure 4-10: GUI of element list and movement option for multiple points

Lastly, several design rules that have proven to be useful should be given to facilitate the entry.

- When drawing lines, always select the point near to the middle as the starting point if possible. This decreases extra work of redirecting translation moving points on multiple line objects.
- Rotating multiple points to generate new reference points in one step can be achieved by selecting not point by point in a point group but applying the full point group in the selection list.
- Always connect straight lines using a hermit spline when they meet near the basic point. This holds true especially for tracks. It is recommended rotating elements in advance to the Scenario variation process by changing a referred angle value in the reference/basic value tab. Often inadequacy within the map design becomes visible. As mentioned, Tracks should always be connected using a hermit spline since seamless transition is crucial for generating a sufficient vehicle handling imitation.
- The naming of elements helps massively in the organization of projects and the recognition of relevant objects, which leads to an increased working speed with more difficult maps.

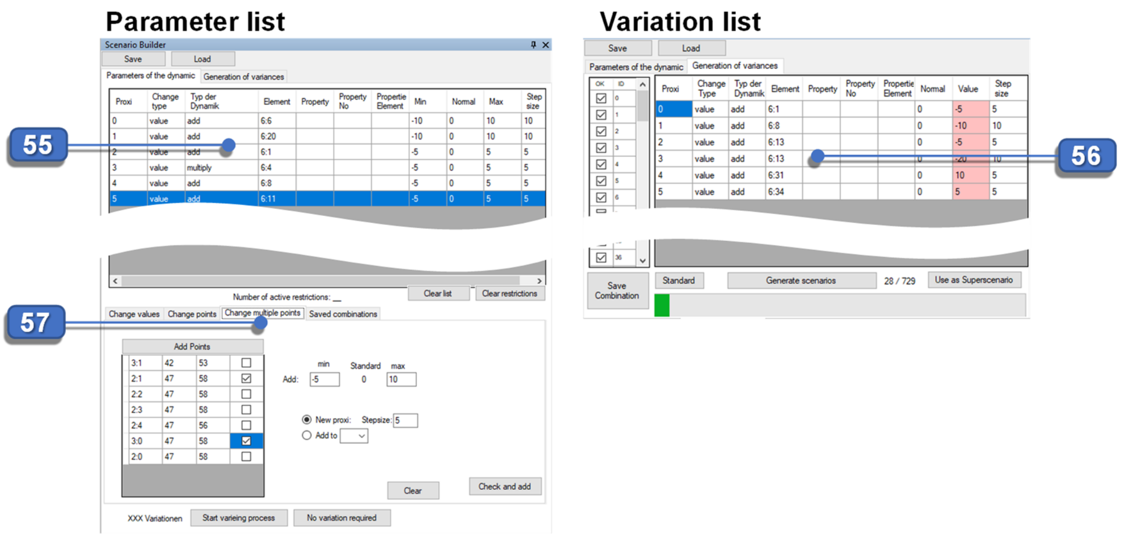
4.3.3 Scenario variation by map diversification

After finishing designing a Superscenario map, only one map is available to train on. This seems very inefficient when considering that simple manipulations, such as the readjustment of objects and the rotation of connected streets at an intersection, lead to a high differentiation of the environment representations.

Thus, a Superscenario variation procedure was programmed to generate Scenarios. This process of variating maps covers two steps. First, variation rules have to be defined, which will then be applied to create possible maps (55). In a second step the user has then to decide which of the resulting maps are used for further simulation sessions and delete those which are not suitable for this purpose (56). Before continuing two terms should be differentiated. A variation defines a modification made to points or values (elements). A combination on the other side, includes multiple variated elements, characterizing all variations made to the

Superscenario to design a Scenario. Both steps required to generate sufficient Scenarios are supported by GUIs which are made visible by clicking the “>” button on the property form. At first, the controls for Scenario variations are shown with a listing of all user-added variation rules in the center (55). The GUI supporting the Scenario variation process is illustrated in Figure 4-11.

In the lower part, the rules can be formulated and changed. In total three different variated concepts are integrated (57).



Proxi	Change type	Typ der Dynamik	Element	Property	Property No	Propertie Element	Min	Normal	Max	Step size
0	value	add	6:6				-10	0	10	10
1	value	add	6:20				-10	0	10	10
2	value	add	6:1				-5	0	5	5
3	value	multiply	6:4				-5	0	5	5
4	value	add	6:8				-5	0	5	5
5	value	add	6:11				-5	0	5	5

Number of active restrictions: __

Add Points			
3:1	42	53	☐
2:1	47	58	☑
2:2	47	58	☐
2:3	47	58	☐
2:4	47	56	☐
3:0	47	58	☑
2:0	47	58	☐

Add: min -5, Standard 0, max 10

(●) New proxi: Stepsize: 5
( ) Add to

XXX Variationen

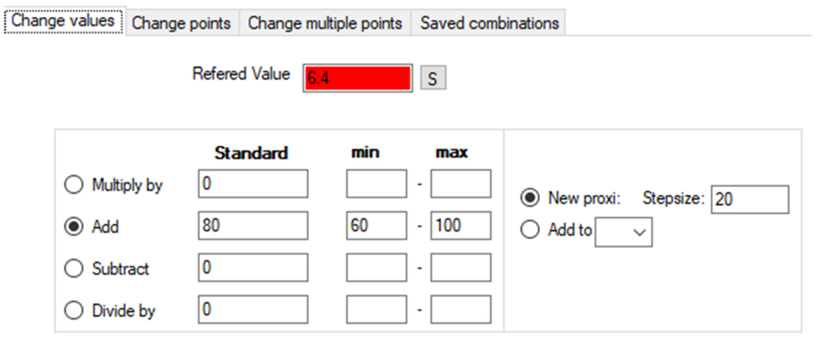
OK	ID
☑	0
☑	1
☑	2
☑	3
☑	4
☑	5
☑	6
☑	36

Proxi	Change Type	Typ der Dynamik	Element	Property	Property No	Propertie Element	Normal	Value	Step size
0	value	add	6:1				0	-5	5
1	value	add	6:8				0	-10	10
2	value	add	6:13				0	-5	5
3	value	add	6:13				0	-20	10
4	value	add	6:31				0	10	5
5	value	add	6:34				0	5	5

28 / 729

Figure 4-11: GUI Scenario variation

Basic and reference values: During the design phase of the Superscenario additional effort was required to define values in order to move obstacles, define the width of streets or set angles for connected street within an intersection Scenario. These values can be uncomplicatedly readjusted with bigger effects on the Scenario by adding a mathematical operator including a minimal and maximal threshold to it. The step-size then enables the software to calculate all resulting variations. Below the step-size textbox, another possible selection referred to as a proxy dependent rule is offered, which will be explained after the listing, since it is contained in all manipulation rules. The elements are illustrated in Figure 4-12



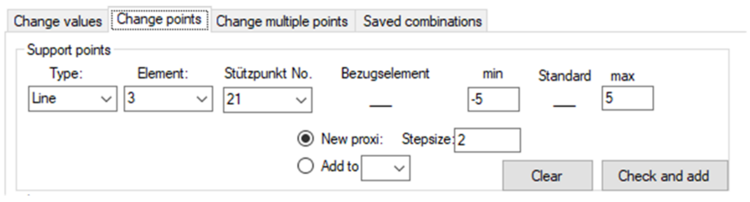
Refered Value: 6:4

	Standard	min	max
( ) Multiply by	0		
(●) Add	80	60	100
( ) Subtract	0		
( ) Divide by	0		

(●) New proxi: Stepsize: 20
( ) Add to

Figure 4-12: Modifying basic and reference values

Location of support points: Characterizing points defining road lines, tracks or environment lines, can be moved on their host line. This allows creating smooth transitions between intersecting road marking lines even if the connected roads are rotated during the Scenario generation. All settings are illustrated in Figure 4-13.



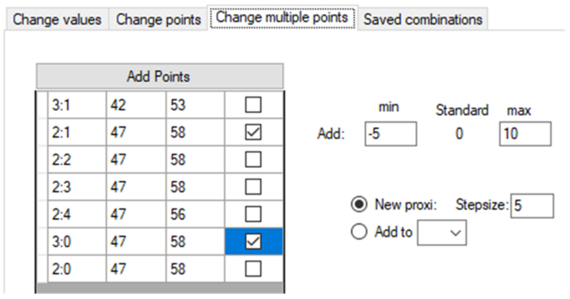
Support points

Type:	Element:	Stützpunkt No.	Bezugselement	min	Standard	max
Line	3	21	—	-5	—	5

● New proxi: Stepsize: 2
○ Add to

Figure 4-13: Location of support points

Moving multiple points on host lines: Instead of varying the position of a single point, as it is shown above selecting and moving points that are located on a host line can be moved together accordingly. This is done by selecting a rectangular area on the designer (similar to the multiple movement features under the element list in the last section) in which all desired points are located. Activated points then are shown in the table. Each row contains a checkbox which turns the direction. This is necessary since tracks and environment lines are orientated in contrast to line markings. When moving elements on a regular two-way street one of two track rows need to be marked as reversed. What differentiates from the multi-point movement feature in the Superscenario, is the ability to vary the range of relocation via upper and lower bound as well as the step-size. The mentioned settings are found in Figure 4-14.



Add Points			
3:1	42	53	☐
2:1	47	58	☑
2:2	47	58	☐
2:3	47	58	☐
2:4	47	56	☐
3:0	47	58	☑
2:0	47	58	☐

	min	Standard	max
Add:	-5	0	10

● New proxi: Stepsize: 5
○ Add to

Figure 4-14: Moving multiple points on host lines

A subject which needs further explanation is the proxy dependent arrangement. When applying rules using the step-size mode, each rule will be executed independently. Having three rules with 20 variations each, in total $20^3 = 8000$ Scenarios are created. Often this procedure will result in many unusable examples with various roads incorrectly intersecting each other. To reduce this overhead, dependent rules have been added which change their focused object property accordingly with the number of steps regarding a leading rule (first rule of the proxy). Instead of modifying each rule separately the software variates proxies instead. With independent rules, including lower and upper bound as well as a step-size, any information necessary is set and a first proxy is defined. To this first proxy other dependent rules can be added by setting the property including the referred proxy ID. During variation the number of possible variations will be calculated by applying the bounding and step-size of the independent rule. The step-size of independent rules is calculated afterward by dividing the range of possible solutions by the computed number of variations from the proxy defining (independent) rule.

Creating variation rules and proxies is simple, managing them, on the other hand, turned out to be challenging. When the variation of Scenarios increases on a large-scale, two major issues have to be faced.

Firstly, required effort is increasing massively when trying to generate a large number of possible variations using a single Superscenario. Rulesets are growing significantly and the number of unusable maps due to the high variation rises heavily when allowing major geometrical elements to change in a large range. Therefore, it is advised to make use of the feature of saving modified Scenarios (after the generation process of the Scenarios) and converting them back as Superscenarios. Then major geometric changes can be applied and reloaded into the Scenario variations.

Secondly, screening of the variated maps and deleting corrupted ones (unchecking their checkboxes) is a time-consuming task, if numbers of maps increase fast. Figure 4-15: Example of a corrupt Scenario. Figure 4-15 illustrates such an unusable Scenario on the left, as well as its positive counterpart on the right.

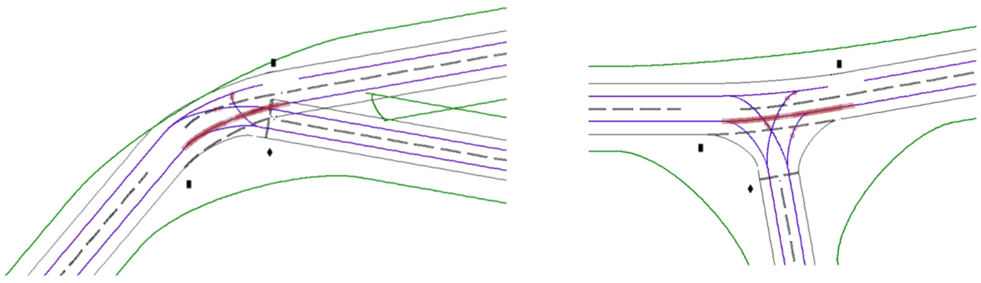


Figure 4-15: Example of a corrupt Scenario

It was found useful not to apply all changes within one single variation process and deleting unnecessary afterward but separately designing rules and eliminating unimportant Scenarios in this resulting subset of solutions. Thus, various subsets of rules including insufficient Scenarios and their property can be saved. When all subsets of rules have been individually designed and their corrupted examples are filtered out, in the last step all rules are reloaded, and a large variation process is applied where all insufficient variations have been deleted in advance. Behind this lays the idea to reduce the dimension of the search space. If a variation was labeled as corrupted, all other variations containing this infeasible subset are also irrelevant. A more in-depth explanation for this subject is given in chapter 4.4.1. At this point it is sufficient to be able to use the tool, as well as to have a basic understanding of its functionality.

A more advanced solution to this problem might be to include constraints to a variation process which decreases false combinations. For instance, adding a minimal delta angle between connected roads in an interception Scenario would reduce the number of overlapping street connections in an intersection Scenario without the necessity of deleting them manually.

Concludingly, Table 4-2 gives some suggestions for different variations that were applied during data generation within this master thesis.

Table 4-2: Suggested list of possible variation types

Lane width	Rough location of traffic lights/signs
Number of lanes of connected streets	Location of stop lines
Angles of connected streets	Ambient location (location of Environment lines)
Location and size of pedestrian crossings	Rough positioning and orientation of objects (Parking vehicles)
Rough hermit spline convexity	Change textures (Angle)
Switching Scenario type between traffic lights, signs and yielding	Add street Scenarios (construction side occupying lanes)

Since each variation results in an own map that needs to be simulated, not every change of an element should result in a new Scenario. Thus, different manipulations including a variation of the position of parking vehicles, road surface markings, traffic signs, and lights, etc. will be made during simulation automatically to improve generalization effects of the trained NN. This subject will be discussed in greater detail in chapter 5.3.1.

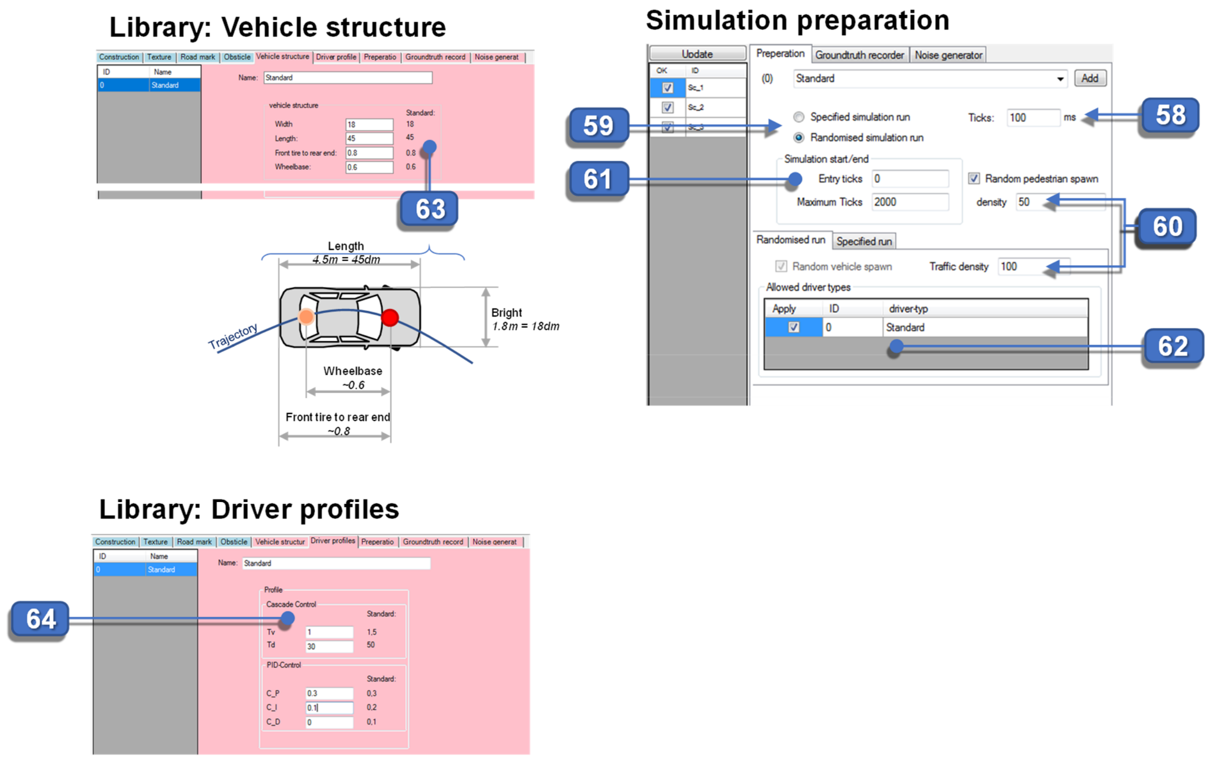
4.3.4 Subscenario calculation by simulation and Ground Truth generation

Before starting a simulation run different settings can be adjusted and finetuned, as the GUI in Figure 4-16 illustrates. To generate an equal understanding of the simulation process, some terms should be explained. A simulation calculates states of vehicles. More precisely the changes within two stages are calculated and then accumulated. The smallest time step these changes are calculated on is referred to as ticks. As a standard they are set to 100 ms (58). Decreasing tick time can lead to longer calculation times. Increasing might lead to larger updates and less accuracy, which results in poor response times and unstable longitudinal behavior of vehicles.

Two basic types of simulation runs are available (59). A specific and a randomized simulation run. The first one definiens the vehicles initial states including its vehicle structure and driver type manually, which allows enforcing a desired traffic situation. The common method used for generating data to train the network in this thesis has been randomized simulations. Obviously, this contains that vehicles, their initial states as well as pedestrians are generated by a randomizer. A vehicle traffic density of 50, 100 and 300 is comparable to rural, inner city and rush hour traffic situations (60). Vehicles can only spawn on the starting points of a predefined route. A support function has been added which interrupts vehicle spawning when another vehicle was spawned previously and is blocking a certain safety corridor. To reduce the amount of empty roads in the generated learning data, the user can enter a number warm-up ticks in which the simulation is calculating vehicle states but does not record them (61). A simulation is finished when a certain amount of ticks has been calculated and recorded.

A vehicle is an object, which is characterized by its appearance (vehicle structure, 63) and its behavior profile (driver type,64). These two details are saved in the library, to allow multiple uses in different projects. A vehicle is defined by its basic dimensions (length and width) as well as the wheelbase and the relative distance from the rear to the middle of the front axle. Both properties are relevant since the vehicle dynamics is based on the single-track-model [47].

Pedestrians spawn randomly on the map within a certain range around the center. Because of time reasons, behavior of pedestrians is kept simple. A route is generated randomly in advance without any guidance. Further work might improve this behavior by adding some guiding points to the map which pedestrians might target with increased probability. It is advised not to increase pedestrian spawn rate significantly since a vehicle decelerates when a pedestrian is on its track.



ID	Name
0	Standard

Name: Standard

vehicle structure

		Standard:
Width	18	18
Length:	45	45
Front tire to rear end:	0.8	0.8
Wheelbase:	0.6	0.6

OK	ID
✓	Sc_1
✓	Sc_2
✓	Sc_3

(0) Standard

Specified simulation run [✓ Randomised simulation run]

Ticks: 100 ms

Simulation start/end

Entry ticks 0

Maximum Ticks 2000

✓ Random pedestrian spawn

density 50

Randomised run | Specified run

✓ Random vehicle spawn

Traffic density 100

Allowed driver types

Apply	ID	driver-typ
✓	0	Standard

ID	Name
0	Standard

Name: Standard

Profile

Cascade Control

		Standard:
Tv	1	1,5
Td	30	50

PID-Control

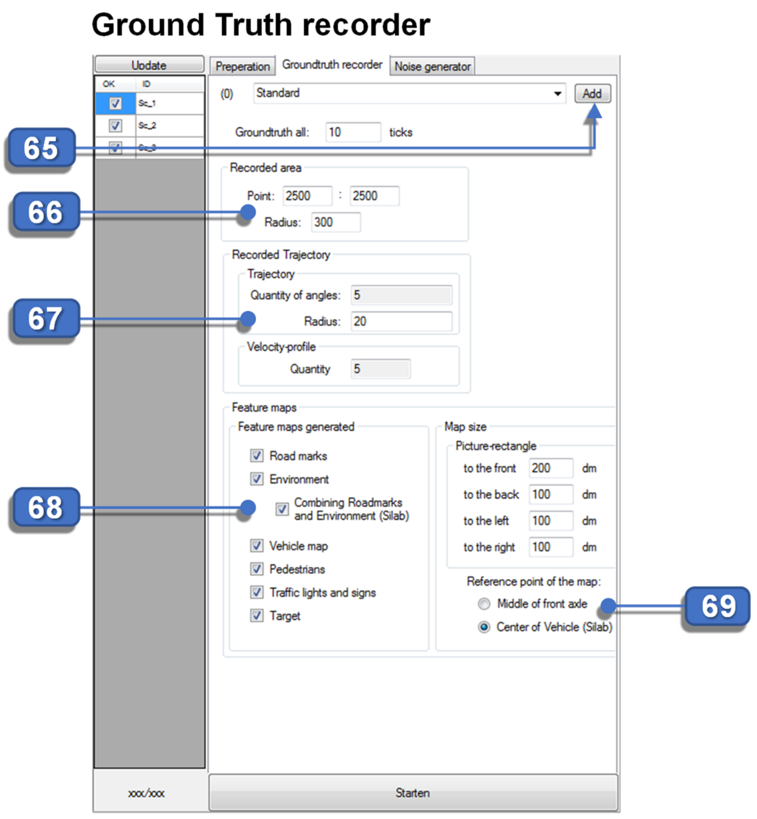
		Standard:
C_P	0.3	0,3
C_I	0.1	0,2
C_D	0	0,1

Figure 4-16: Simulation preparation, vehicle structure and driver profiles

Data recorder

After a simulation has finished, the results need to be saved so they can be used for training the network later on. Just as for the simulation preparation, data recorders can repeatedly be used thus making it more comfortable having a library entry for it (65). Focusing on the central inner area of a Scenario, data from entering roads should only be represented on a reduced level. Since the mentioned data is highly similar, the network could not learn any more complex behavior from data containing mostly single vehicles driving on a straight road (66). Furthermore, data with no steering would be overrepresented, and therefore their number needs to be decreased.

As mentioned before, the required data consists of the environment representations and Ground Truth information. The latter is further subdivided into a velocity profile and a trajectory path including a fix radius and the trained steering angles to define their support points in a polar coordination system (67). Depending on the input information an agent has in exploit phase, it might be necessary to have different ensembles, sizes and centers of environment representations for training. The simulation then can be started and observed by a progress counter (68, 69). The data recorder GUI is presented in Figure 4-17.


Ground Truth recorder

Update | Preperation | Groundtruth recorder | Noise generator

OK	ID
✓	Sc_1
✓	Sc_2
✓	Sc_3

(0) Standard — Add

Groundtruth all: 10 ticks

Recorded area
- Point: 2500 : 2500
- Radius: 300

Recorded Trajectory
- Trajectory
  - Quantity of angles: 5
  - Radius: 20
- Velocity-profile
  - Quantity 5

Feature maps
- Feature maps generated
  - ✓ Road marks
  - ✓ Environment
    - ✓ Combining Roadmarks and Environment (Silab)
  - ✓ Vehicle map
  - ✓ Pedestrians
  - ✓ Traffic lights and signs
  - ✓ Target
- Map size
  - Picture-rectangle
    - to the front 200 dm
    - to the back 100 dm
    - to the left 100 dm
    - to the right 100 dm
  - Reference point of the map:
    - ○ Middle of front axle
    - ● Center of Vehicle (Silab)

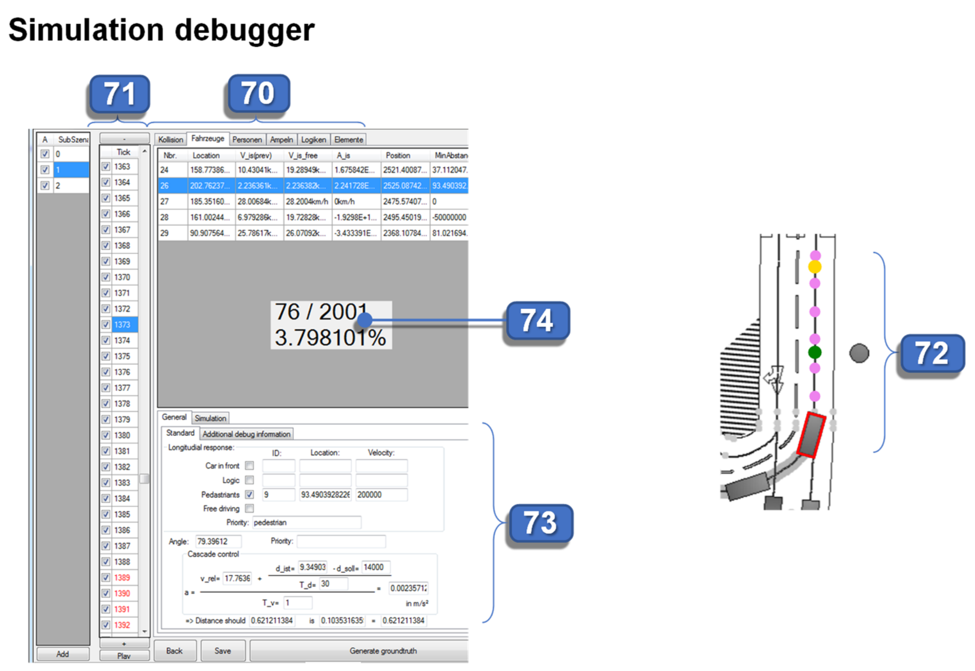
xxx/xxx — Starten

Figure 4-17: Training data generator

4.3.5 Simulation results and debugging

A simulation can be started by two different modes. By advancing in the property checklist in section 4.3 the debug mode starts which visualizes the simulation results. A second mode (server-client) is recommended for simulating large sets of Scenarios for which the simulation environment was developed. This will be introduced in the next section. As debug mode implies further changes in the software, but also new Superscenario maps should be tested in this mode before applying them in server-client mode. This is recommended for new maps since often more complex Scenarios require fine-tuned placement of logic elements as expected to function correctly.

In the upper part of the center all simulated items (vehicles and pedestrians), as well as occurring collisions during the simulation are listed (70). To achieve an easier failure detection every tick is visualized by showing the full environment representation including any object which was placed on it manually or is related to the current simulation step (71). For a selected vehicle, all trajectory path points, as well as the optimal target are visualized. Information about the planned velocity is shown in the group-box below the vehicle list (72). When starting a simulation, a counter shows the process of the current session including finished ticks (74), as it is illustrated in Figure 4-18.



A	SubSzena
☑	0
☑	1
☑	2

	Tick
☑	1363
☑	1364
☑	1365
☑	1366
☑	1367
☑	1368
☑	1369
☑	1370
☑	1371
☑	1372
☑	1373
☑	1374
☑	1375
☑	1376
☑	1377
☑	1378
☑	1379
☑	1380
☑	1381
☑	1382
☑	1383
☑	1384
☑	1385
☑	1386
☑	1387
☑	1388
☑	1389
☑	1390
☑	1391
☑	1392

Nbr.	Location	V_is(prev)	V_is_free	A_is	Position	MinAbstan
24	158.77386...	10.43041k...	19.28949k...	1.675842E...	2521.40087...	37.112047
26	202.76237...	2.236361k...	2.236382k...	2.241728E...	2525.08742...	93.490392
27	185.35160...	28.00684k...	28.2004km/h	0km/h	2475.57407...	0
28	161.00244...	6.979286k...	19.72828k...	-1.9298E+1...	2495.45019...	-50000000
29	90.907564...	25.78617k...	26.07092k...	-3.433391E...	2368.10784...	81.021694

76 / 2001
3.798101%

Longitudial response:

		ID:	Location:	Velocity:
Car in front	☐			
Logic	☐			
Pedastriants	☑	9	93.490392822€	200000
Free driving	☐			

Priority: pedestrian

Angle: 79.39612 Priority:

Cascade control

d_ist= 9.34903 - d_soll= 14000

v_rel= 17.7636 +

T_d= 30

a = = 0.0023571:

T_v= 1

in m/s²

=> Distance should 0.621211384 is 0.103531635 = 0.621211384

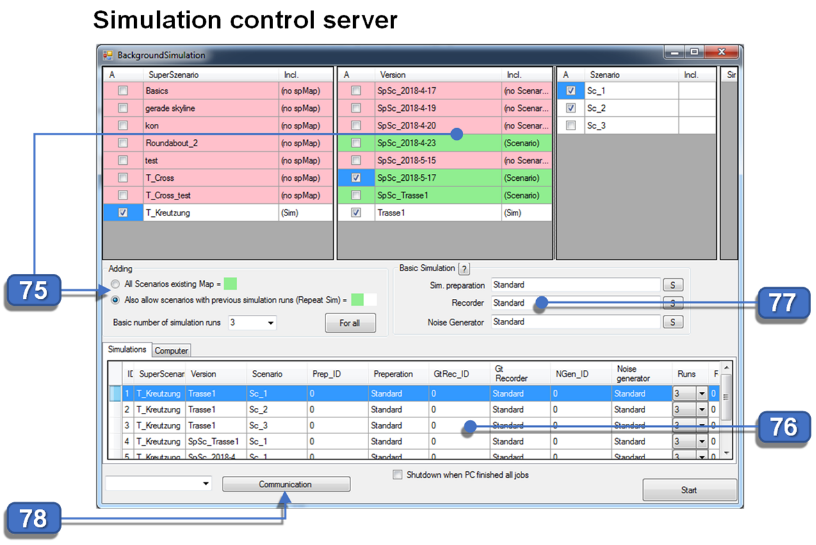
Figure 4-18: Debug visualization

4.3.6 Simulation scaling via server-client solution

Hence simulation and Ground Truth generation are computationally expensive, a solution must be found to increase the speed of the process. Unfortunately, the costliest part is the data generation. Microsoft‘s *.net* framework uses GDI+ to draw polygons and lines on bitmaps which is very computational intensive compared to a plain simulation.

After reducing the code from unnecessary computation, each simulation with 2000 steps takes up to 30 seconds, Ground Truth generation more than 30 minutes on a commodity PC. Considering 25 Superscenarios with 100 Scenarios each, a total simulation time of 52 days is required to only computing labeled training data. In the Server GUI, covered in Figure 4-19, all Simulations can be started and managed during runtime. This provides a single access point for the user to execute simulations across multiple desktop PCs. In the upper half a color-coded list containing all Superscenarios, Scenarios, and Subscenarios. If a Scenario already has been utilized for simulations, it is colored in white. Green means that no simulations have yet been started. Red illustrates that currently no Scenario map is available for simulation purposes (75). Before adding simulation jobs to the scheduler (76), simulation settings must be selected from the library (77). In the scheduler list, all information is presented regarding the progress of individual jobs and the responsible client. Furthermore, the server is capable of shutting down PCs via simple communication when required (78).

Arguments can be added when starting the simulation software via command line or including a separate argument to an existing link file in Windows. They allow starting the software either in server or client mode directly. This simplifies the starting procedure and mode selection process when running the software on multiple computers in a network. The following arguments are integrated. `\Server` and `\Client` directly open the server or client GUI. When adding `\Client start \Ground_path C:\Somepath…` the client GUI, as it is illustrated in Figure 4-20, will be presented as the initial form with the ground path already predefined and the connection set up (79). Using the last argument, the server can immediately start sending jobs to the client which is already in waiting mode. To start multiple instances on several PCs more easily a txt file can be added with the commands stated above. This is recommended when starting the client software on a large number of PCs.



A	SuperSzenario	Incl.
☐	Basics	(no spMap)
☐	gerade skyline	(no spMap)
☐	kon	(no spMap)
☐	Roundabout_2	(no spMap)
☐	test	(no spMap)
☐	T_Cross	(no spMap)
☐	T_Cross_test	(no spMap)
✓	T_Kreutzung	(Sim)

A	Version	Incl.
☐	SpSc_2018-4-17	(no Scenar...
☐	SpSc_2018-4-19	(no Scenar...
☐	SpSc_2018-4-20	(no Scenar...
☐	SpSc_2018-4-23	(Scenario)
☐	SpSc_2018-5-15	(no Scenar...
✓	SpSc_2018-5-17	(Scenario)
☐	SpSc_Trasse1	(Scenario)
✓	Trasse1	(Sim)

A	Szenario	Incl.
✓	Sc_1	
✓	Sc_2	
☐	Sc_3	

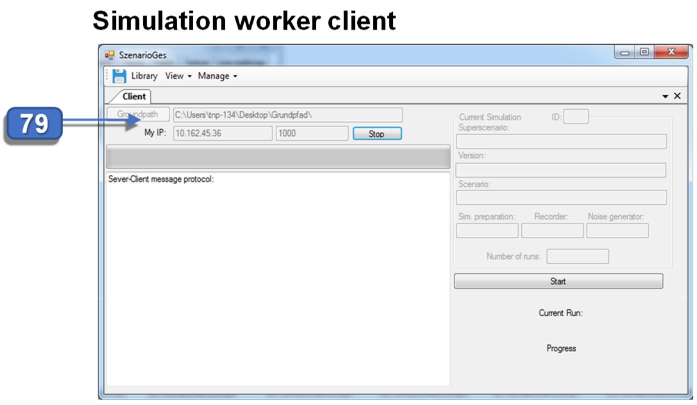
All Scenarios existing Map = 🟩
Also allow scenarios with previous simulation runs (Repeat Sim) = 🟩
Basic number of simulation runs: 3

Sim. preparation: Standard
Recorder: Standard
Noise Generator: Standard

ID	SuperScenar	Version	Scenario	Prep_ID	Preperation	GtRec_ID	Gt Recorder	NGen_ID	Noise generator	Runs	F
1	T_Kreutzung	Trasse1	Sc_1	0	Standard	0	Standard	0	Standard	3	0
2	T_Kreutzung	Trasse1	Sc_2	0	Standard	0	Standard	0	Standard	3	0
3	T_Kreutzung	Trasse1	Sc_3	0	Standard	0	Standard	0	Standard	3	0
4	T_Kreutzung	SpSc_Trasse1	Sc_1	0	Standard	0	Standard	0	Standard	3	0
5	T_Kreutzung	SpSc_2018-4	Sc_1	0	Standard	0	Standard	0	Standard	3	0

☐ Shutdown when PC finished all jobs

Figure 4-19: GUI of the Server



Groundpath: C:\Users\tnp-134\Desktop\Grundpfad\
My IP: 10.162.45.36 1000

Sever-Client message protocol:

Current Simulation ID:
Superscenario:
Version:
Scenario:
Sim. preparation: Recorder: Noise generator:
Number of runs:

Current Run:
Progress

Figure 4-20: Client GUI

4.4 Major features of the simulation environment

After describing the overall process of generating and running a simulation, different hidden and more complex features should give a deeper inside into the functionalities of the simulation environment. It needs to be mentioned that these explanations will not focus on a deep programming level. Instead, their basic principles and functionalities should be pointed out to give the reader an idea, of why they are important and how they work.

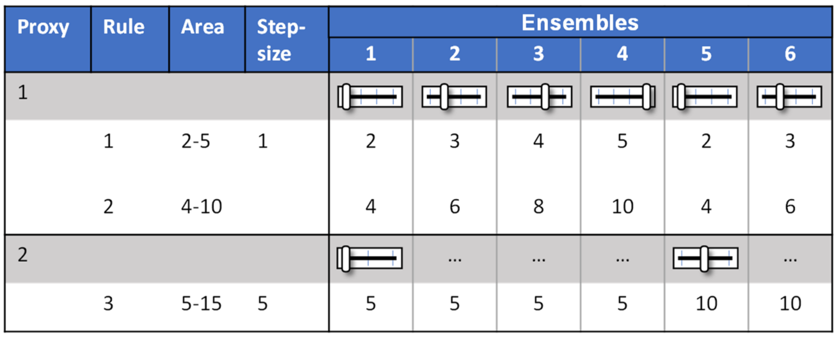
As was implied in the previous section, the developed simulation environment uses two types of variation processes to increase data output from a comparatively small number of basic Superscenarios. Namely, different street course, time-related traffic simulation and logging of multiple vehicles. The developed solution for the first process (varying street courses) will be explained in chapter 4.4.1. A more detailed presentation on vehicle and pedestrian behavior within the traffic simulation is given in section 4.4.3. Failure detection is a key to high data quality. Especially, crashes and corrupted situational data must be detected and eliminated to avoid training an insufficient unintentional behavior of the Neural Network. Measures to achieve this goal are shown in section 4.4.4. Map based data generation is subject of the last explained feature in chapter 4.4.5.

4.4.1 Scenario variation process

As mentioned before, the simulation software has been written relying on Microsoft's *.net* Framework, which is an object orientated programming language. Thus, all listed elements are defined as objects including various properties. These objects can be easily duplicated and modified by applying a predefined combination of rules.

Before getting into detail a few notations should be explained to have unified phrases for the same subjects. A rule in this issue is a single entry in the change list, containing a minimum, maximum value and a step size, depending on the type of manipulation. Since often the user is not interested in a full independent ruleset, but having some rules dependently changed, proxies have been added to the software. A proxy contains a single or multiple rule and is applied in the main recursive variation function to generate all combinations of Scenarios. Minimum, maximum value and step size combined can be seen as a scale. Each proxy is considered as a dimension building up a possible solution space. The possible solution space contains all generated maps through grid-based rule application. These solutions are elements that have been selected and considered as suitable maps to create training data from.

At this point, the functionality of proxies should be explained. As mentioned above, a proxy is the basic unit that is employed in the changing process defining the states of a property within a Scenario. Each proxy is characterized by the number of interpolation steps (changes) it is applying to its containing rules. A rule defines the object (property) to change the lower (min) and upper (max) bound. Each proxy also includes a leading independent rule which contains the step-size. This is used to calculate the number of interpolating steps which is then transferred to compute the step-size of other included dependent rules. A proxy is changed by updating its state which then results in changing the properties that are included in the attached rules accordingly. A proxy can be imagined as a single slider that shifts through all connected rules and adapting them consistently. The following example in Figure 4-21 illustrates this.



Proxy	Rule	Area	Step-size	Ensembles					
				1	2	3	4	5	6
1									
	1	2-5	1	2	3	4	5	2	3
	2	4-10		4	6	8	10	4	6
2					...	...	...		...
	3	5-15	5	5	5	5	5	10	10

Figure 4-21: Slider illustration for proxies

Rule 1 and 2 depend on each other because both share the same proxy. Rule 1 is the guidance for the proxy since it declares the step size. The final third rule is independent for illustration purposes. All possible ensembles containing any variation of the proxy and independent value are generated.

Even with applying proxies and dependent rules the possible solution space can contain an unmanageable amount of solutions. When having ten different proxies each with five steps the total number sums up to $10^5 = 100{,}000$ single possible solutions. Some regulation tools are necessary to reduce this number to an agreeable level. Since each proxy defines a dimension, it is useful to change the feasible solution space for any proxy separately and combine them afterward. This is done by dimension reduction. Instead of applying all modifications and building a large combination, which then has to be screened into single step batches of rules, they are utilized and filtered separately. As Figure 4-22 illustrates, rule 1 and 2 are combined and checked as a batch (orange area contains insufficient solutions). Rule 2 is a separate batch with a single dimension. By combining both rules with their insufficiently marked areas, the solution space is significantly reduced without additional effort.

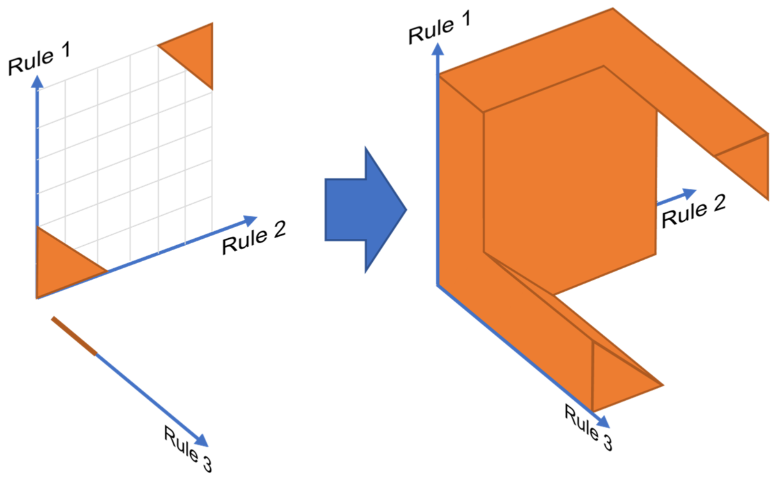
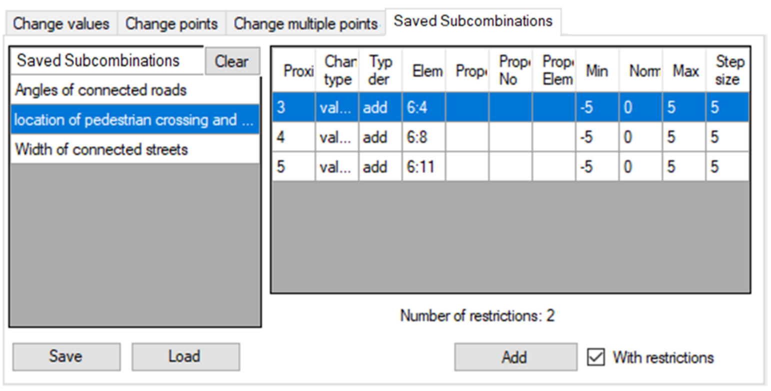


Figure 4-22: Separated filtering (left), combination (right)

In the Simulation environment, this has been integrated by saving and loading Subcombinations. Each of these contains a small subset of parameters which was varied. Subsequently, all corrupted maps (variations) have been sorted out. For each deleted map a restriction is added, containing their set of parameters. These variations are then saved as a Subcombination, including any variation and restrictions. All mentioned steps are iteratively performed for all combinations of parameters which depend on each other. In the last step, all Subcominations are loaded into the variation list by using the Subcombination manager GUI which is illustrated in Figure 4-23.



Saved Subcombinations
Angles of connected roads
location of pedestrian crossing and ...
Width of connected streets

Proxi	Char type	Typ der	Elem	Prop	Prop No	Prop Elem	Min	Nom	Max	Step size
3	val...	add	6:4				-5	0	5	5
4	val...	add	6:8				-5	0	5	5
5	val...	add	6:11				-5	0	5	5

Number of restrictions: 2

☑ With restrictions

Figure 4-23: GUI for managing Subcombinations

The manager then checks and deletes variation during a combined variation process, if an imported restriction concerning a parameter subset is met.

4.4.2 Simulation process and network computing

A simulation is built upon several predefined simulation steps, with each step changing the states of all available vehicles and pedestrians. A tick is the time-difference between each simulation step. Changes of states are calculated relative to this tick-time. This section describes each computational step that is resulting in a state update of the simulation and any including dynamic element (vehicles, pedestrians, and traffic light status). The repeating process is illustrated in Figure 4-24.

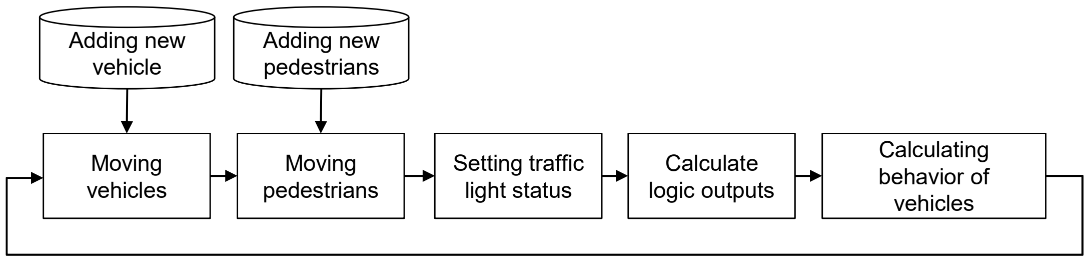


Figure 4-24 Computational steps during a simulation update

A simulation update starts with a query in a filled vehicle database, including each vehicles ID, route and entry simulation step. When a vehicle's entry step is equal to the current state number, it will be added to a second *active* list, containing all vehicles that are currently on the map. To ensure, that no vehicle is blocking the new entrant's route points near the spawn point are checked for existing vehicles. When it is blocked, the vehicle will not be added to the *active* list.

The movement of the vehicles is performed at the beginning stage of a simulation step. If a new vehicle is spawned, its velocity is initialized to the desired track speed of the beginning of the route. In case the vehicle already has been active, a behavior change might occur at this point resulting in an updated vehicle state. The traveled distance is then calculated considering a constant speed or acceleration depending on the driving mode a vehicle is assigned to. For a more profound insides into vehicle behavior generation read the upcoming section 4.4.3.

Similar to vehicles, pedestrians spawn when their initialization tick is reached. Pedestrians spawn randomly within the inner data-generator circle to reduce extra computations for objects which do not have any effect on the training data. At spawn time the route of a vehicle will be calculated in advance, by combining randomized pieces of movement passages (including heading change and velocity). Pedestrians tend to change their direction slightly but also have the capability to redirect themselves with larger delta heading angles, which is rarely executed. Additionally, it must be noted that pedestrians neither react to static objects or pedestrian crossings nor to approaching traffic to minimize the risk of an accident, which should be trained as a vehicle's responsibility.

In the next stage logic elements are switched according to the current simulation step, followed by calculating stopping point shifting controlled by logic elements. As been mentioned, a logic element requests if a vehicle is located on an assigned range of observed track points and switches stopping points on different tracks accordingly.

After vehicles are activated (spawned), their behavior is computed considering the interaction with third-party vehicles, pedestrians, logics and basic track-related desired velocity. The reasons for calculating the vehicle's behavior at the end of a simulation step is due to the fact that movement is time-related. All calculations of a single simulation step are intended to happen parallel and instantaneously with no time elapsing. Only between simulation, the time of a tick is passing by which can be used to compute the changes in velocity, location and heading of dynamic objects.

4.4.3 Vehicle behavior and movement

The movement of a vehicle can be split into a longitudinal and lateral proportion [48]. This reduces the effort of determining and computing a car's position, state, and behavior. The center point of the front axle defines the location of the vehicle throughout the simulation. As been mentioned, vehicles move on routes which consist of multiple tracks. The center point always matches with a point on a track. To increase preciseness of calculation, not integer points (tolerance of 10cm) are chosen for locating vehicles, but a *double* value which describes the position of focused objects as the progress relative to the length of the route they are moving on.

Streets provide gross transversal guidance in the shape of a lane. The centered track line will be used as a guiding trail for the vehicle, defining its lateral location.

Longitudinal movement holds some more significant challenges. In order to simulate a car's motion, the basic concept of adaptive cruise control (ADC) was implemented. Velocity is affected concerning three different possible driving states.

Free driving mode: If a vehicle is not interacting with objects in its environment, the free driving mode is activated. In this state a vehicle follows the desired speed predefined by the tracks velocity profile. Since this profile is calculated by interpolating user generated support points no gaps, spikes or discontinuities are expected, which meant that no controller is necessary. Unfortunately, it cannot be ensured that no jumps occur within transitions between tracks. Therefore, a basic PID-control has been implemented to balance these discontinuities.

Following mode: When following a vehicle, two primary tasks need to be accomplished, starting with detecting a preceding vehicle and adjusting the ego vehicle's velocity to it. For detection purposes, shifting between routes and tracks becomes relevant. To determine preceding objects the future route is observed to locate potential blocking vehicles. Since different routes may contain the same track, it is necessary to check any route for vehicle location, direction, and velocity.

Each vehicle is recognized by the track point representing the center of the rear axle instead of the front axle which is used to calculate its location, velocity and behavior. This was implemented since many collisions occurred hence standing vehicles were not detected by the following ones. This was due to the fact that the preceding and following vehicle were assigned to different routes which shared the same track. At the location where both routes are branching the preceding vehicle had to decelerate and stop with having the characterizing front axle point on the new track but the rear one remaining on the old (shared track). Since in previous development states only considered checking for the front axle a collision was unavoidable.

In the vehicle following mode, the speed is regulated by a cascade control used in ADC systems shown in Formula (4-1). It is considering the difference in velocity ($v_i - v_{i+1}$), as well as the distance to the vehicle in front ($d_{is} - d_d$). Two factors are used to tune the desired responsiveness regarding the mentioned velocity τ_v and distance delta τ_d to follow the vehicle keeping a desired safety distance [49, p. 891].

$$a_{i+1} = \frac{v_i - v_{i+1} + \frac{(d_{is} - d_d)}{\tau_d}}{\tau_v} \tag{4-1}$$

A representation of the cascade control circuit is given in Figure 4-25.

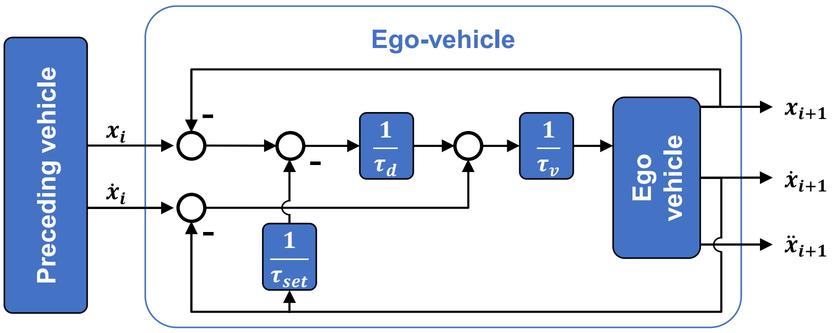


Figure 4-25: Representation of the cascaded control circuit [49, p. 891]

Reaction to pedestrians has also been achieved with the cascade control. Instead of only determining current velocity the perpendicular distance of pedestrians is taken into consideration for calculating vehicle’s velocity. If a pedestrian is expected to cross the road (a lateral distance of one and half of the vehicle’s width) the vehicle will decelerate and stop. Within a perpendicular range of twice of the vehicle width, near passing is considered with a maximal velocity of 10 km/h.

Logic elements: As mentioned, logic elements are used for implementing a set of basic road regulations into the simulation. Logics can be considered as stopping points on a track which can be switched on or off, when a predefined condition is met. These conditions include occupancy requests for certain other tracks to implement yielding or stopping and time-triggered events set by traffic light states. At the beginning of the development it was intended to apply the cascade controller on both vehicles and logics. Unfortunately, no parametrization was found which showed plausible braking behavior for interacting with moving vehicles and static stopping points alike. Either the vehicle stopped several meters before the desired location, or it collided with slowly moving vehicles. To address this problem, a simple linear velocity control is triggered when interacting with logic elements decelerating the vehicle to stop at the predefined location.

After calculating the resulted velocity considering all three driving modes, the mode is executed which results in the shortest traveled distance. The relocation of the vehicle will be performed at the beginning of the next simulation step.

4.4.4 Failure detection within the simulation

Checking and detecting collisions is crucial for not passing corrupted examples into the Training data set. To teach the NN a valid behavior, incorrect examples must be identified and discarded at during runtime. Besides, collision detection is used as a debugging check for the simulation framework as well. Collision detection is held simple to reduce computational costs. It is [50, p. 6] achieved by determining line interception. Each vehicle is characterized by its rectangular bounding box. Pedestrians are defined by an eight-point circular shaped polygon. The crash detection algorithm is implemented by calculating the cross product for two-dimensional vectors. Each outer line of a polygon will be checked if two objects are nearby (>50 dm). Applying the cross product then checks if point A1 of an object is located on one side and the other point A2 on the different side of object B's line $\overline{B_1B_2}$. When both points A1 and A2 are located on different sides delimited by line $\overline{B_1B_2}$ a crash has been detected. [51, p. 41]

After implementing collision detection errors have been found and corrected. For instance, there was a little chance that multiple cars were spawned on one route concurrently. Because both cars were close, no visual feedback was given that both cars collide during the whole run. After detecting these ongoing collisions, the spawn algorithm has been extended to also check for progress of other cars on same track peace and only releases vehicles on a route when a minimum distance to others is provided.

4.4.5 Data-generation for Neural Network training

CNNs require two-dimensional matrices (images) as inputs. In this thesis, the NN's seven channels (road markings and environment, moving vehicles, pedestrians, traffic signs, and target, as well as current speed) are used for predicting a trajectory.

Before focusing on the realization of the separate maps a more general topic about map size and resolution should be discussed. With an increasing size, the required computing power leads to rapidly rising learning durations. A high resolution can increase the precision of the map content but reduces the possible visible range. A compromise was found, containing a map size of 300x200 pixels with a resolution of 0.1 m per pixel. To increase the front viewing distance, the center of the focused vehicle is set back by 50 pixels, resulting in a 2to1 ratio in favor of the front vision. The decision was taken, considering limited sight and a speed limitation of 50 km/h within urban areas. Later, the resolution for vehicles was doubled to reduce the probability of abrupt braking maneuvers. When considering an application in real traffic, a larger map size is recommended for safety reasons (taking into account the ten second response time for the driver in a level 4 vehicle). Nevertheless, due to limited computational capabilities, the 300x200 pixel maps have been considered sufficient for the proof of concept.

Line maps: Since data about static lines and objects are part of the Scenario, the environment can be rotated and centered concerning the location and heading of the focused ego vehicle. The background is blackened, all lines are colored in white. Only contours of obstacles and environments are plotted, due to the later used canny edge detector filter generating the testing data in the applied SILAB testing-environment, which just prints white contours as well (chapter 6.3).

Moving objects (vehicles and pedestrians): Another relevant subject to be discussed is the visual appearance of moving vehicles and pedestrians. This information has to be transferred to the NN and processed with minimum extra computational effort. As shown, cars and pedestrians are represented by their location, size (bounding box), orientation and velocity. Mapping of velocity and heading has been achieved by the following concept. Basically, an 8-bit image allows 256 shading levels. When limiting the velocity to a permitted range of 50 km/h (Speed limit in closed villages and inner cities in Germany), a fixed function can convert the speed into pixel brightness. Since the velocity of pedestrians does not reach these levels and heading direction should be recognizable for the network their maximal speed (pixel brightness = 256) ends at 10 km/h for a fast-moving person. Since the background (no vehicle on this spot) is determined as ground level 0, an unmoved car must not be shaded in 0 in order to be recognized as a valid object. No clear rule has been found. Therefore, an own simple rule has been developed. To recognize a car, its bounding box is filled in the value of at least

128. This results in a range of 128 units for a velocity field of 0 – 50 km/h and sufficient resolution of approximately 0.4 km/h which can be considered as an adequate precision. To restrict speed beyond 50 km/h in the simulation environment, the maximal desired velocity of the tracks has been regulated. Since maximal speed only results from allowed track velocity in free driving mode, with logics and ACC following mode just reducing it, no further adaptions were necessary to the software. The center of gravity is the center point for directional representation. Starting at this point, a line is crossing the bounding box/circle in the direction of current yaw. The resulting intersection defines the point of maximal brightness. Initially, the center of the front wheel's axle was used since they directly stick onto the track/route and front-wheel steering is assumed. Since the testing of the developed movement planning solution is performed on a traffic simulation software (SILAB), which considers the center of mass as coordinate origin, the setup of the own software was changed accordingly. The appearance of vehicles and pedestrians are illustrated in Figure 4-26.

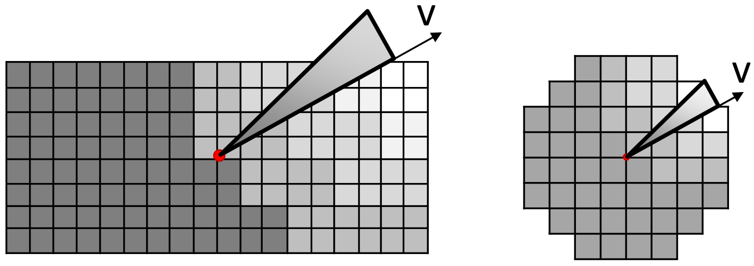


Figure 4-26 Illustration of the shading of a vehicle and pedestrians

A workaround, which does not require timeline information (like velocity) packed into the channel input is to use Recurrent NNs. They use a set of channel inputs and determine the speed and direction by their own. Even if it might be the more elegant approach to this problem, it enhances the complexity of the simulation. As shown in the section about the simulation process, Subsimulations are required to prevent the network to consider information which the network does not have, to a specific point in time.

Traffic signs and lights: To held map creation simple, basic bitmaps were drawn onto the blackened map to illustrate the presence of a sign. Traffic light states are represented by different gray shadings. For further explanations please see Table 3-1.

Target: The interface connecting the movement planning algorithm and the desired destination given by satellite navigation or operator input is provided by the target. To simplify calculations a target state is characterized by its target value ranging $[-1, \dots, 1[$. Within the simulation, the target value is computed by calculating the point of intersection between the route a vehicle is currently driving on and a 300x200 rectangle moved together with the ego vehicle. The value is then calculated via the schematic illustrated in Figure 4-27 (left side).

It must be considered that the given target only is a rough approximation of the desired location. Hence reduced precision of operator input including transition delay, or delocalization due to slight changes in street locations (construction sides), veering and passing obstacles, as well as GPRS errors, may emerge. Thus, instead of training on a fix value randomness must be added. A randomizer applying a Gaussian distribution (standard deviation of 0.3) was integrated for this purpose. A simple summation then manipulates the target value with the error, as shown in Figure 4-27 (right side).

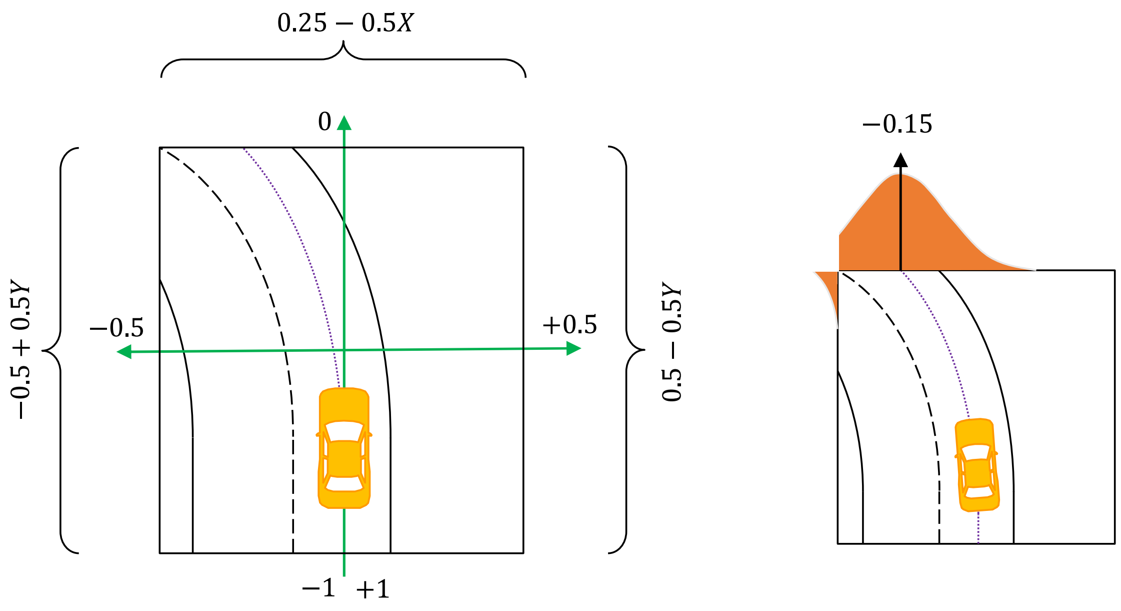


Figure 4-27 Target value generation

To conclude the tuple of input maps for the network the current velocity is provided by adding a monotone shaded 300x200 pixel map. The value is calculated during batch loading directly while training the network. Since it is advised to have normed channel inputs reaching between -0.5 to 0.5 (float types) the current speed (0 to 50 km/h) is linearly mapped considering the desired range. This reduces storage on the hard drive by reduced calculation time.

4.5 Results and generated maps

During three weekends 26 Desktop PCs from the institute have been employed as clients including a server PC. On each Client PC six instances were run parallelly. During this period 25 Superscenarios, including 5332 Scenarios have been simulated, which resulted in a total number of 70,145 files and 7,014,500 data points (each containing six environment representations).

Since storage on university network drive is restricted to 40.000 files, it was required to build packages of 20x5 sets with a 300x1200 size container including the mentioned six 300x200 maps. A broad range of possible required information is saved onto a CSV file. Each line represents a data point, including the ego vehicle's map location in the map package, current velocity and Ground Truth (trajectory). In Figure 4-28 some examples of generated maps are illustrated in the 20x5 grid.

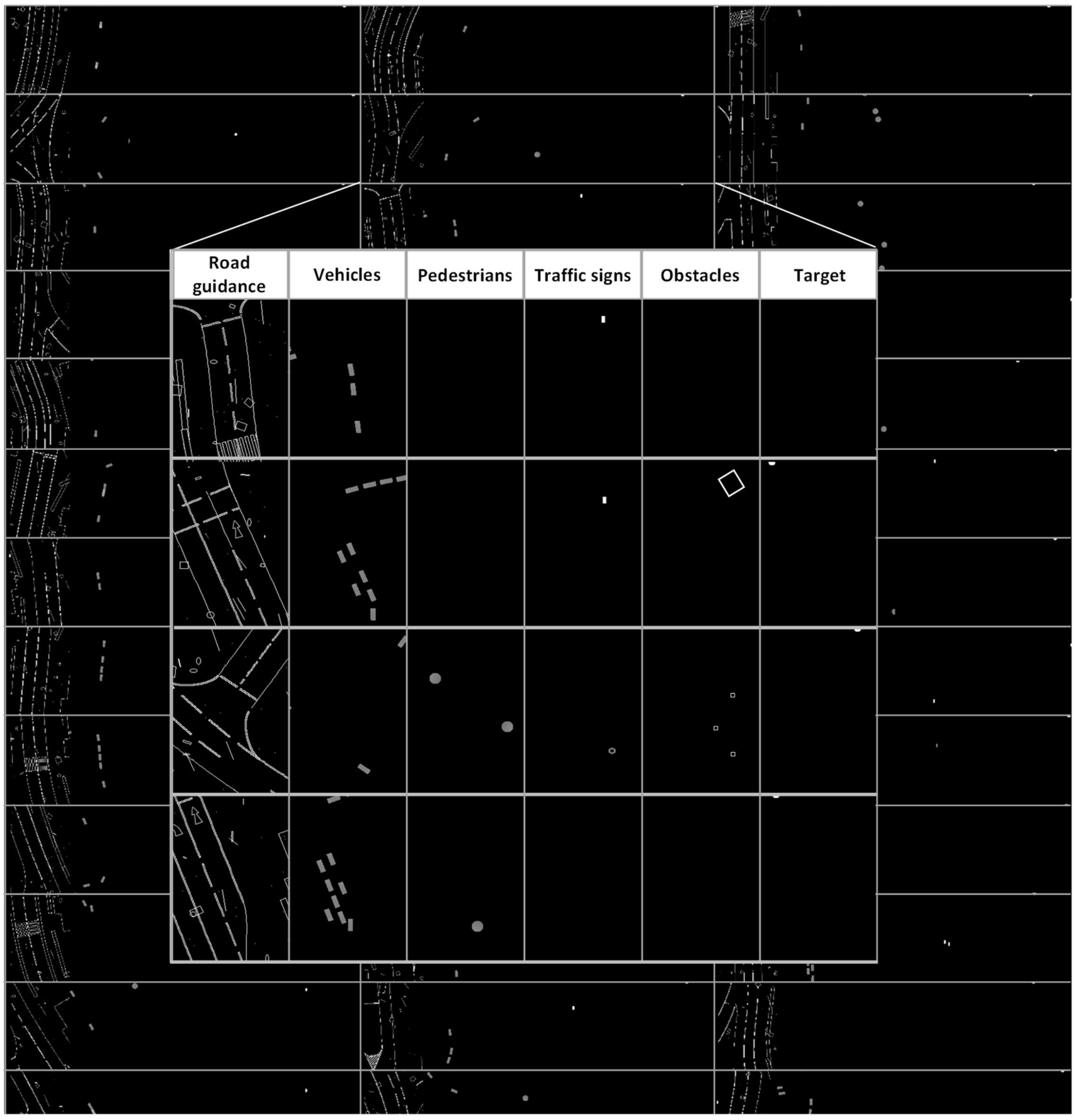


Figure 4-28: Exemplary data set generated by the simulation environment

The CSV file has been extended with data required for further Error-analysis containing information about the current focus of the vehicle (third-party vehicles, pedestrians, logics, traffic signs and lights, etc.) as well as surrounding objects and collisions. The combination of these map packages and CSV data tables are then used for training the NN, on which the next chapter will focus on.

5 Neural network design

In this chapter the undertaken training and performance improvement steps regarding the CNN solution are explained in greater detail. This includes a roadmap which has been developed and applied during this phase of the master thesis project and is explained in section 5.1. To increase performance using more advanced methods in section 5.2 an evolutionary algorithm which has been implemented to optimize HPs is introduced. Furthermore, in section 5.3 the Error-analysis solution including its results regarding developed NN and agent improvement approaches have been described and discussed.

5.1 Roadmap for learning procedure and performance improvements

Neural Network engineering is a complex often undetermined and unordered procedure, involving various steps. Each step itself often requires comprehensive insides into Deep Learning. This section should give a guideline and a short overview of performed tasks for selecting, training, validating and improving NNs during the development process. Figure 5-1 illustrates the entire roadmap.

Basically, designing and developing an intelligent agent was achieved by three repeatedly performed procedures. Starting with the training phase, including an architecture and Hyperparameters a model is trained. After and during training sessions its evaluation error has been checked to approve increasing performance and selecting the best network for the testing phases. In the last step, these good performing models have been tested on data provided by the SILAB traffic simulation software.

To speed up the repeating training and evaluation processes, integrated data management, version management, as well as training sessions, a second software was written to support the user and automatize recurring tasks.

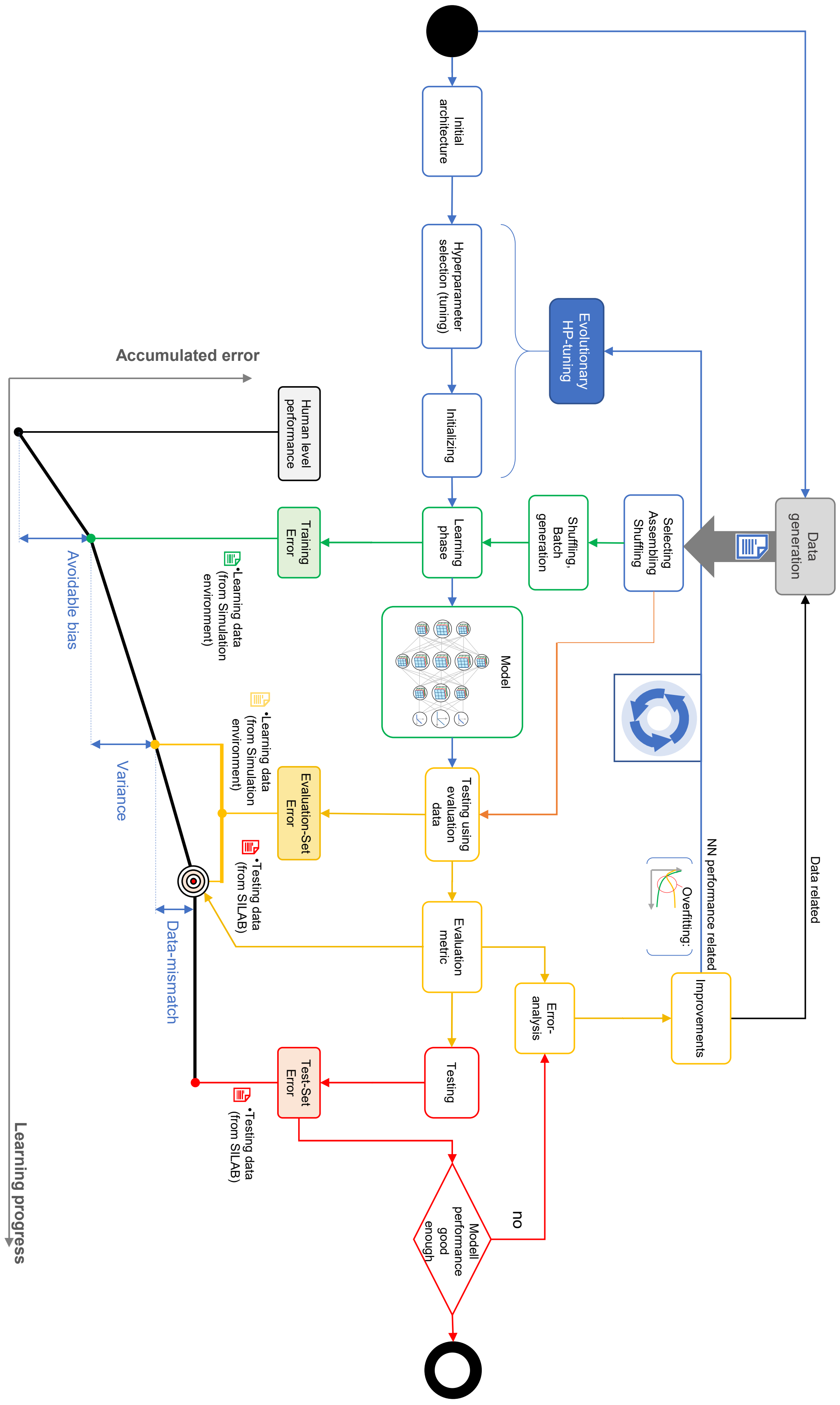


Figure 5-1: Applied roadmap for NN training, testing and improvement

5.1.1 Initial architecture and network adaption

Starting with the project a sufficient network type and an underlying architecture has been determined. Often new Deep Learning project adapt previously trained NNs (also called transfer learning) [52, 53]. In this project, no such network existed which resulted in engineering a new approach.

The result concerning this step, a concept for movement planning for autonomous and teleoperated vehicles, has been presented in chapter 3. This includes the definition of inputs and outputs, a primary network type as well as a first initial network size and other Hyperparameters, as they are listed in the next section.

5.1.2 Hyperparameter-tuning

Different Hyperparameters have a significant impact on the performance of networks. Therefore, it is crucial to choose them carefully. Some of the most relevant CNN HPs are listed in Table 5-1 including their common specification range [54, 55, p. 3, 56, 57].

Table 5-1: Most important HPs

Hyperparameter	Common range of specifications
Kernel Initialization	Random
Activation functions	ReLU, Arctan, Sigmoid
Optimizer	Adam, Adamax (including various setups)
Learning rate	0.1 to 0.001 depending on the activation function and learning state
Number of layers	Any positive integer number
Number of filters or nodes	Any positive integer number
Dropout rate	0-50%
Batch and mini batch size	Any positive integer number
batch normalization	0/1
Stride	Any positive integer number

Fortunately, a wide range of research papers exists to provide guidance for HP-tuning [54, 58, 59]. Instead of explaining the effect of every single HP and listing rules for their parametrization, which already can be found in many papers, the overall search procedure should be explained in the next paragraphs. Basically, four types of HP optimization exist, including grid and random search as well as Bayesian search and heuristics rules.

Applying a basic grid search means to calculate the performance of millions of different HP combinations. This solution gives a 100% chance of finding the best performing network, but one can easily see, that with average training and validation duration for a single CNN of 5 to 10 hours the summed calculation time would exceed the work time of the master thesis by far.

Random search does not suffer from a necessarily long runtime due to a fix number of HP combinations. However, this method does not have to find the HP setup resulting in a global optimum regarding NN performance. Furthermore, no improvement must be encountered during the optimization process [60, p. 285]. This implies that no indication is given whether the found performance might has converge.

Bayesian search and the neuroevolutionary approach can be seen as a compromise between both mentioned methods [61]. A Bayesian search takes previously gathered knowledge into consideration to shrink the search area before a next search iteration starts. This results in faster improvements, but depending on the applied type, this concept can also lead to saturation in a local minimum.

The evolutionary algorithm (also called generic algorithm) replicates the mechanisms of biological optimization and replacement techniques. Compared to Bayesian search, this approach includes mechanism to counter this issue of early saturation by randomly changing HPs (mutation). Therefore, the neuroevolutionary approach has been chosen over the mentioned ones and will be presented in greater detail in the next section after the roadmap has been presented.

5.1.3 Initialization and organization

Initialization includes debugging, preparing as well as organizing training data and scheduling training sessions to reduce overhead as far as possible. For these steps a software has been written which supports the user to generate NNs from Keras templates automatically, organize data and training sessions as well as document training progress.

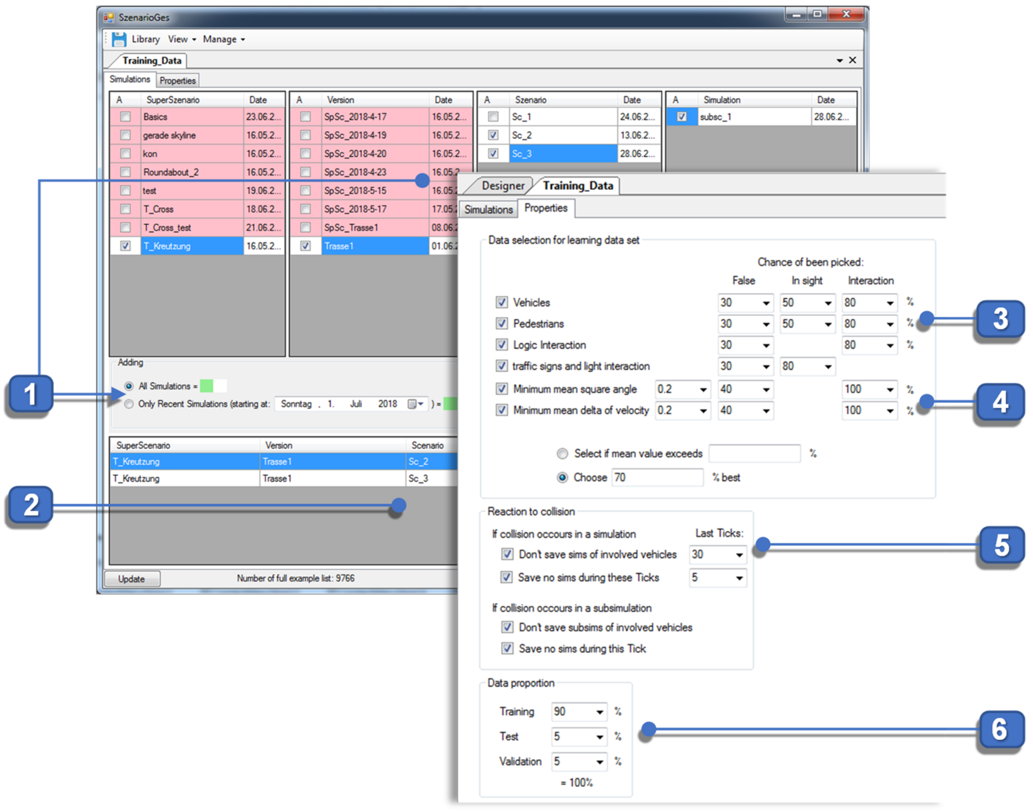
5.1.4 Data generation

In chapter 4 a software for the generation of training data, containing environment representations and Ground Truth data has been presented. In combination with the Scenario variation and included traffic simulation a source for high quality and quantity training data has been developed. With an integrated server-client feature allowed parallel computation within a single organized session by scattering simulation on a larger amount of commodity desktop PCs. On university drives a limit of 40.000 files exists, which is easily exceeded by simulating some Superscenarios. Therefore, environment representations had to been packaged to bitmaps of 5by20 tiles each.

5.1.5 Selection and assemble of training data

Often, when coding smaller CNN projects, pictures are loaded directly from all folder directories into the network. When considering the size of the training data including the necessary file structure and the characteristics of this project a more professional solution is required. Therefore, a data assembler has been developed to rank, shuffle, and combine data elements for NN training and evaluation. Its GUI is illustrated in Figure 5-2.

Training data picker



A	SuperSzenario	Date
☐	Basics	23.06.2...
☐	gerade skyline	16.05.2...
☐	kon	16.05.2...
☐	Roundabout_2	16.05.2...
☐	test	19.06.2...
☐	T_Cross	18.06.2...
☐	T_Cross_test	21.06.2...
✓	T_Kreutzung	16.05.2...

A	Version	Date
☐	SpSc_2018-4-17	16.05.2...
☐	SpSc_2018-4-19	16.05.2...
☐	SpSc_2018-4-20	16.05.2...
☐	SpSc_2018-4-23	16.05.2
☐	SpSc_2018-5-15	16.05.2
☐	SpSc_2018-5-17	17.05.2
☐	SpSc_Trasse1	08.06.2
✓	Trasse1	01.06.2

A	Szenario	Date
☐	Sc_1	24.06.2...
✓	Sc_2	13.06.2...
✓	Sc_3	28.06.2...

A	Simulation	Date
✓	subsc_1	28.06.2...

Adding

[✓ All Simulations =] 🟩
Only Recent Simulations (starting at: Sonntag , 1. Juli 2018) = 🟩

SuperScenario	Version	Scenario
T_Kreutzung	Trasse1	Sc_2
T_Kreutzung	Trasse1	Sc_3

Number of full example list: 9766

Data selection for learning data set

Chance of been picked:

		False	In sight	Interaction	
✓ Vehicles		30	50	80	%
✓ Pedestrians		30	50	80	%
✓ Logic Interaction		30		80	%
✓ traffic signs and light interaction		30	80		
✓ Minimum mean square angle	0.2	40		100	%
✓ Minimum mean delta of velocity	0.2	40		100	%

Select if mean value exceeds %
[✓ Choose] 70 % best

Reaction to collision

If collision occours in a simulation — Last Ticks:
- ✓ Don't save sims of involved vehicles: 30
- ✓ Save no sims during these Ticks: 5

If collision occours in a subsimulation
- ✓ Don't save subsims of involved vehicles
- ✓ Save no sims during this Tick

Data proportion

Training	90	%
Test	5	%
Validation	5	%
	= 100%	

Figure 5-2: Data assembler

The basic selection process is held very similar to the server GUI in the server-client solution in chapter 4.3.6. In the upper half, all available Scenarios are presented with appropriate color-coding (1). Red shows that no suitable training data is found in this Scenario. White illustrates that training data is available, but at least one directory includes files which are older than a selected date. Green list entries present data that are newer, respectively. This has been introduced since during the fast-growing and changing project no proper version management was suitable. When selecting a training data file, it is added to the lower list. A counter then determines and shows the number of elements within all comprising data selections (2).

As been mentioned, data diversity is a critical issue in Deep Learning. During Scenario generation, one essential element has been to variate maps in order to increase variance. In the data generation procedure during traffic simulation, the data recorder only considers vehicles which are located in a focus range of the critical Scenario element (For instance a 40m range for an intersection). This helps to minimize training data containing vehicles that are merely following a basic road. To further decrease redundancy, a function has been added which allows ranking and deleting data. The user can design a point system to rate data. This system includes reaction states of vehicles regarding third-party vehicles, pedestrians, traffic lights and logic elements. The point system contains three rating elements including direct reaction, visibility and no visibility concerning the related object granted with a decreasing amount of point. In addition to the response states points are given for changes in the steering angles and velocity profile to increase the importance of dynamic vehicle reactions during training (3). Furthermore, it is possible to delete examples which are linked to a collision within the simulation to ensure high-quality training data (4).

After each training example has been graded, an algorithm ranks the data and deletes weaker training samples. The threshold is calculated by selecting the proportion of remaining training data. Additionally, the threshold can also be defined by choosing a minimum amount of points for each data sample directly (5).

Subsequently, the data will be shuffled and rearranged in new 20x5 map packages. Apart from the image shuffling process the connected vehicle state data (including Ground Truth etc.) requires some additional focus. To keep data organization as simple as possible, the data assembler generates two CSV lists. The first file (deep_learning.csv) is applied for training and testing. Thus, it only contains Ground Truth and location of the connected maps (file name of 5x20 map package and coordinates of the exact environment representation group). To simplify loading and organization in Python each row contains the information of 100 examples connected to one map package.

To allow proper Error-analysis in subsequent steps data backtracing is required. Thus, a second CSV file lists each data point with information containing its original Scenario and current vehicle states.

During usage it was found necessary to execute the data assembler on multiple computers since shuffling, loading and saving images is associated with long runtimes (each shuffled 5x20 environment representation package requires approximately a minute to compose on available regular desktop PC). To reduce the number of training data collections that are created by a single desktop PC and parallelly do not impair randomness, the user can define a fix number of Subscenarios (simulations) to be included which is then shuffled and processed. Hereafter the software is started on multiple desktop PCs, applying the full range of Subscenarios while only composing up to 500 of them in a single run.

Subsequently, in case it is required to generate a broader dataset, a merger is added to the software which automatically combines multiple selected training sets to a single larger ensemble which can be used as training and evaluation set.

5.1.6 Shuffling and batch sizing

Data shuffling in deep learning is a crucial element to accelerate the optimization process [50, p. 6]. Shuffling in projects applying image packages is more complicated than single image shuffling. Loading a full package to export a single training data during training is insufficient due to the necessity of additional (already restricted) computation capacities. Thus, the shuffling process was split into two separate procedures to find a compromise between reduced computational costs and high randomization of the training data. The first one has already been explained in the last subsection including shuffling and assembling training data from Subscenarios including the creation of new image packages. This process has been finished before learning begins. A generator supplies the training process with necessary data to update the model. One task of the generator is to load the images and ground truth to order and buffer them in batches. If a batch size of 100 is requested only one package is loaded, instead of loading several packages and excluding single examples. This reduces computations and decreases learning times massively. Thus, shuffling of these packages had to be done in advance during package assembly. Elements within one package are rearranged within the generator which does not result in an increased effort since the image is already loaded in memory. The full process of data shuffling is illustrated in Figure 5-3.

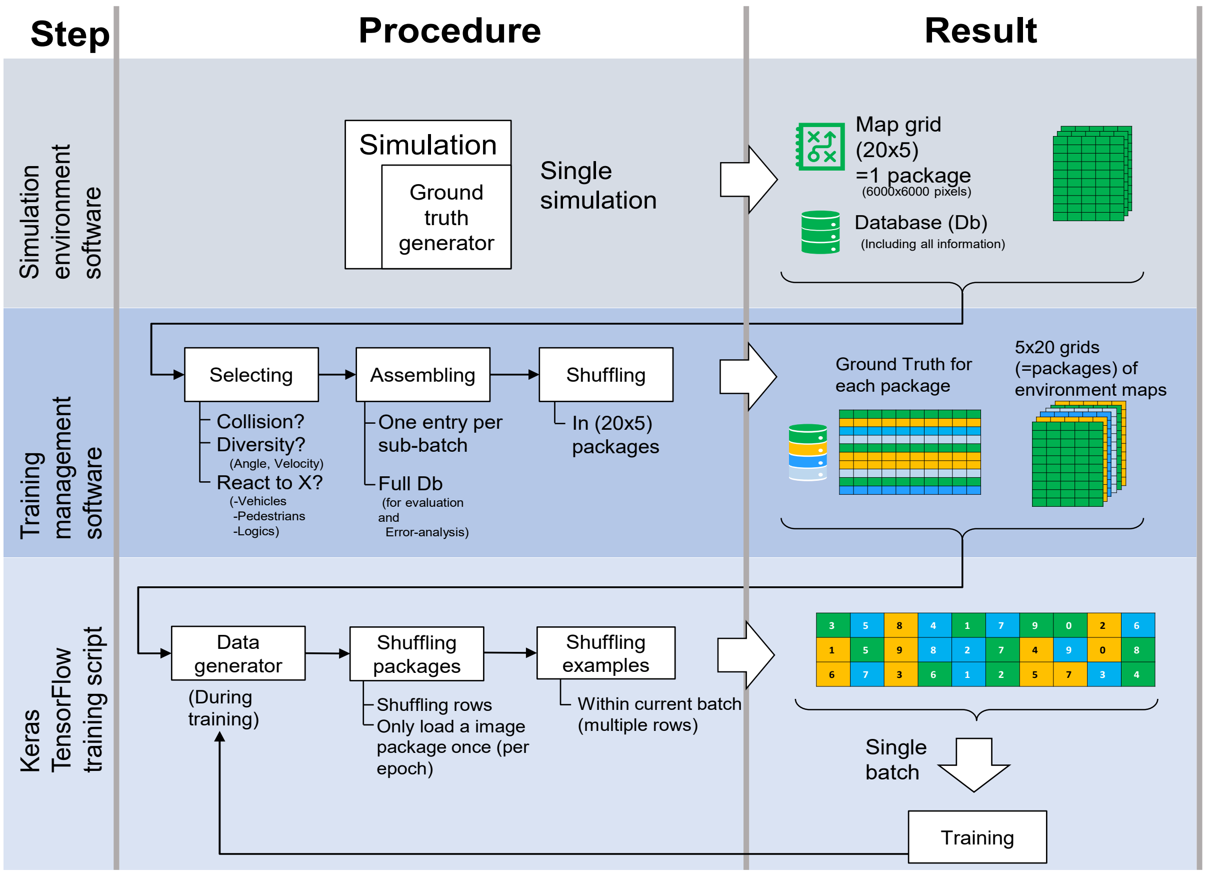

Step
Procedure
Result
Simulation environment software
Simulation
Ground truth generator
Single simulation
Map grid (20x5) =1 package (6000x6000 pixels)
Database (Db) (Including all information)
Training management software
Selecting
Assembling
Shuffling
Collision? Diversity? (Angle, Velocity) React to X? (-Vehicles -Pedestrians -Logics)
One entry per sub-batch
Full Db (for evaluation and Error-analysis)
In (20x5) packages
Ground Truth for each package
5x20 grids (=packages) of environment maps
Keras TensorFlow training script
Data generator
(During training)
Shuffling packages
Shuffling rows
Only load a image package once (per epoch)
Shuffling examples
Within current batch (multiple rows)
Single batch
Training

Figure 5-3: Data shuffling process

5.1.7 Learning phase

After a model including its HPs and training data is provided the training process can begin. To decrease programming effort, Keras has been used as a front-end library for TensorFlow. This enables easy model generation and multi-GPU support. Apart from the process steps that are described in this chapter, this documentation should also explain the major obstacles and errors that one need to take into consideration and react accordingly if necessary to improve NN performance.

Avoidable bias: The bias error describes the gap between Ground Truth and predicted values calculated while weight optimization during the training phase. This offset can further be subdivided into avoidable and unavoidable bias. The last error either results from inefficiencies of learning algorithms procedures which is the objective of fundamental research or occurs since no complete knowledge and Ground Truth concerning a subject is available. Minimizing the avoidable bias is more relevant in this thesis since its detection and correction is usually manageable during machine learning projects focusing on an application level. When encountering a high bias, a first step, to begin with, is to check the training data for inconsistencies. Nonsystematic errors may lead to an increasing bias since the network is disimproving during one update and correcting this failure in the upcoming iterations [62]. Utilizing updates on mini batches can balance the appearance of those erroneous updates.

Systematic errors in training data, on the other hand, may not have a direct effect on the training error but massively reduce NN performance during the application phase. Since they usually become visible during evaluating NN performance against evaluation data, this issue is discussed in the upcoming section 5.1.9. Another crucial step regarding the training data is to analyze whether the data input contains any information necessary to enable an appropriate prediction by the NN. This type of error can only be minimized during concept development. The avoidable bias heavily relates to the human performance level. If a human is capable of mastering a task, sufficient training data can be generated containing the problem (NN input) and the related solution (Ground Truth), vice versa.

To ensure that no information is missing, several colleagues have been given multiple input data points, a NN will be provided with, to draw a trajectory on a spreadsheet and discuss possible velocity changes. During this process, it was found necessary to increase the input by adding the current velocity of the ego vehicle. Lastly, a high bias may also result from a maladaptive training setup, including choosing an inadequate learning rate, high dropout rates or false L2 regularization and optimizer parametrization. Reducing the amount of training data points while increasing the number of epochs a NN is trained on may reduce the avoidable bias but results in other significant problems, as described in section 5.1.9.

5.1.8 Machine Learning model

A trained model is characterized by its weights, biases (parameters) as well as an amount and type of layers including various activation functions etc. (HPs). Usually, one architecture trained on the same data within the same number of epochs results in different parameters and performance. This is mainly due to random initialization and data shuffling by the data generator. A performance improvement, therefore must be significant to be considered relevant.

If a computing resource is not available during a complete NN training session or the full range of data is not available at the beginning of the project, recurrent learning phases including transfer learning are a possible option to train larger networks with many parameters. After finishing training on multiple epochs, a model’s parameters, weights, and biases can be saved as a h5-file and eventually restarted and continued in future sessions, if needed.

5.1.9 Testing using evaluation data set

After training a model, a first evaluation is required to ensure improvements in performance.

Evaluation error: The evaluation error is defined by the difference between the training and evaluation/validation set error. This error might often result from two inadequacies.

Either overfitting occurs which is an indicator of a too complex model (NN) which is finding patterns which only exist in the training data. To resolve this issue either the amount of training data can be increased, dropout regularization and early stopping should be applied, or a smaller model has to be designed [44, p. 115]. Unfortunately, decreasing evaluation performance over epochs might not always indicate the necessity of reduced network size. Occasionally, an even larger network shows increasing performance on the evaluation set. A possible reason might be that the smaller network was not capable of finding the best pattern to represent the underlying information given by the data, which requires more parameters, only a larger network can provide. However, a clear ensured explanation cannot be given for this phenomenon. To address this problem, HP-tuning needs a level of randomness which the evolutionary algorithm, introduced in section 5.2 will provide (mutation).

During this project, multiple networks have been trained to find a network which does not overfit applying the evolutionary algorithm for HP-tuning (section 5.2.1). Additionally, due to the development of a simulation environment data could have been generated quickly in sufficient quantities.

The second reason for an occurring evaluation error might be incorrect Ground Truth which does not match with the training data. When applying corrupt data for NN training the network will most likely find a hypothesis to match and predict the given erroneous ground truth. Even if the training procedure is executed correctly a bad performance is to expect since prediction relies heavily on data quality. To ensure high quality input the training data many debug sessions were undertaken to improve the simulation environment. Furthermore, the data assembler also detects and deletes collisions to minimize the risk of training a weak performing hypothesis.

5.1.10 Testing on application data

The primary task of performance optimization in this project is not to find a setup that enables best NN results, but to increase the performance (or intelligence) of a intelligent agent. Optimizing a NN is not an end in itself but rather a mean to develop a good performing AI. Deep Learning is about finding patterns in predefined data. AI performance does not only depend on the discovered patterns but also on the properties of underlying data and is mainly related to the application of the software.

Test error and data mismatch: As the term data mismatch implies, errors might also occur, since features or elements that are present in the data which the NN has to perform on during practical application is not integrated into training and testing data. To decrease the risk of suffering a data mismatch issue, a NN should ideally be trained on a mix of training and real application data [62]. If no large set of this data is available during the training phase, NNs should also be tested on application data and relevant mismatches have to be considered by an Error-analysis to adapt training data generation process accordingly. During training stages, only limited testing data from SILAB could have been applied for training since the agent was still under development. This has been the reason why a more sophisticated Error-analysis also considers the evaluation on some (50) SILAB application map examples. Since no Ground Truth was available, the trajectory has been printed directly on the maps. These included special situations including pedestrian crossing, intersections with traffic lights as well as others.

5.1.11 Error-analysis

During an Error-analysis the developer searches for fields of improvement concentrating on errors occurring on the validation set and in the testing environment. Since training data is shuffled and relevant information is divided and saved in multiple files a software has been written to support the user finding errors and chances for improvements. Found errors can result in data and network (training) related performance enhancements. As an essential and often explained example, the subject of overfitting can be detected by perceiving a rising evaluation set error, while determining decent training improvements. This problem might be solved by adapting Hyperparameters including downsizing network architecture, regularization or applying methods like early stopping or increasing the training data set. This procedure is considered relatively trivial. However other more complex improvements require a more profound insight knowledge about the underlying application. Therefore, this subject is outsourced and described in section 5.3 in greater detail.

5.2 Evolutionary approach to Hyperparameter tuning

If complex non-linear backbox models must be optimized without having any more in-depth information about rules and logics involved in the improvement process, the application of a genetic algorithm is a sufficient way to address this issue. This is an iterative optimization method derived from the evolutionary processes of nature [63, p. 22]. Described below is the genetic algorithm Using binary combinations to solve non-linear problems. This has been limited since the present chapter is only intended to provide a short introduction to the processes of the algorithms. In order to make the operations comprehensible, it is necessary to briefly discuss some of the terms which are essential for understanding the algorithm [64, p. 15].

An individual refers to a possible solution of the algorithm. It is characterized by a binary sequence (chromosomes). The chromosomes are composed of individual genomes. These consist of genes which are the smallest unit of information [63, p. 35]. The structure of chromosomes is illustrated in Figure 5-4.

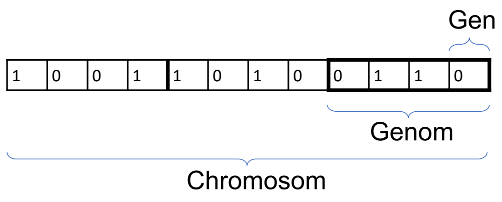


Figure 5-4 Structure of a chromosome

The fitness function (optimization function) measures the quality of employed individuals [65, p. 7]. The individual with the best fitness also corresponds to the currently best solution of the optimization problem [64, 11, 31]. A population can be considered as a group of individuals. One generation is the totality of all individuals within a single iteration.

Beginning with the initialization phase, several individuals are created containing a random binary sequence [65, p. 8]. Subsequently, the iterative optimization process then starts, as shown in Figure 5-5 [64, p. 25].

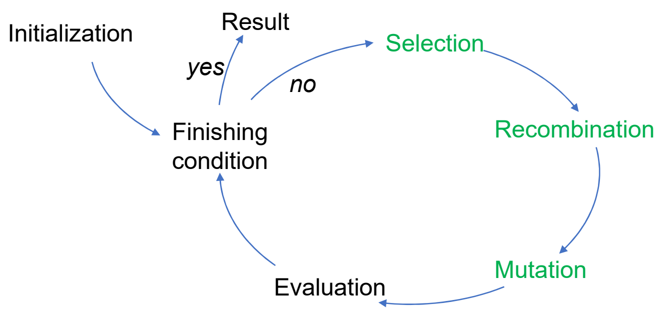


Figure 5-5: Computational steps of evolutionary algorithm [64, p. 25]

Variation and combination processes alter individuals within a population [65, p. 8]. The goal of these operations is to drive the optimization process forward. The task is not to improve single individuals by directed actions. Instead, the focus is on the optimization progress of the entire population [9, p. 11]. The modification of individuals is usually applied randomly. This increases the fitness of some solutions, while others might lose performance to some extent. The first step is the pairing selection, in which individuals with the goal of crossing are put together into subgroups. In the recombination step (also called crossover) random sections of parents' chromosomes are selected and joined together for creating a new individual (children) [65, p. 8]. Figure 5-6 shows a recombination process involving two parents and two resulting children [66, p. 4].

A:	0101	0011	0110	1010
B:	1010	0000	0001	0010
C_{new}:	0101	0011	0001	0010
D_{new}:	1010	0000	0110	1010

Figure 5-6: Exemplary recombination step

In the mutation phase, random changes are made to elements of the population. This is often limited to small adjustments, as shown in Figure 5-7 [65, p. 8] .

A:	0101	0011	0110	1010
Mutation:	0000	0000	0001	0010
A new:	0101	0011	0111	1000

Figure 5-7: Exemplary mutation step

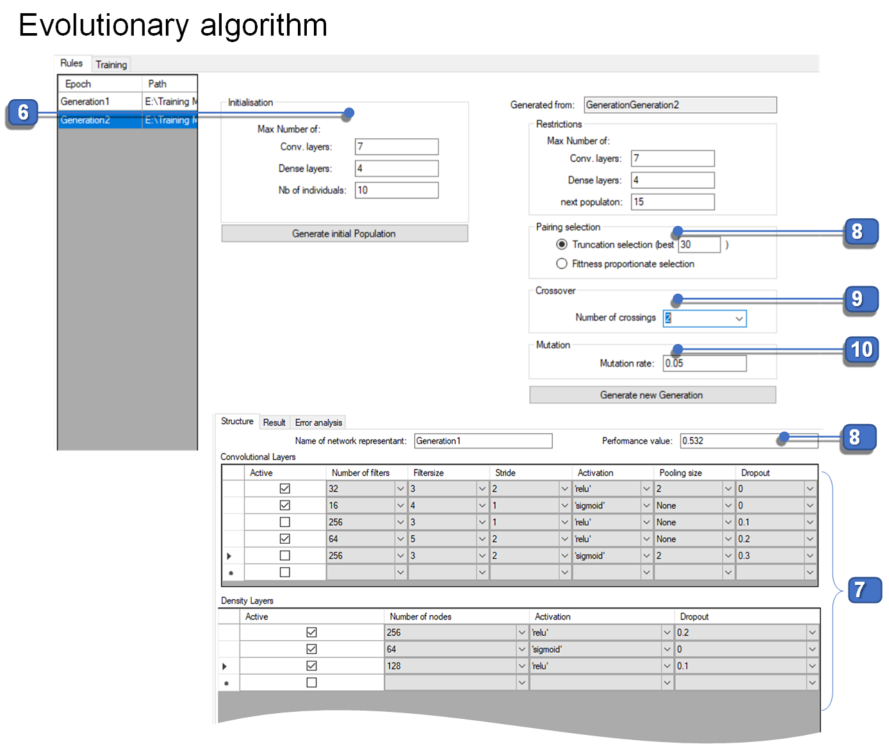
The scheduling condition determines whether the iterative optimization algorithm is continued or canceled. If a satisfactory solution has been achieved or a fixed number of iterations have been computed, the procedure ends [64, p. 25].

5.2.1 Adoption for CNN Hyperparameter tuning

To adapt the evolutionary optimization method to HP tuning to increase performance some changes have to be made in contrast to the previously discussed basic binary method. In this section an implementation (strongly inspired by [67]) will be explained by going through the steps of the evolution algorithm schematic. Starting with the basics, an individual represents an NN, including a full set of HPs as its chromosome. A genome is equivalent to a layer. Genes describe a single specification of an HP in a specific layer. A package of NN trained and evaluated in a single iteration is referred to as the population of a generation.

The fitness value describes a NN performance on a standardized, not changing test set. Just as the testing data, the epochs and initial weights should also be considered static over all individuals during the execution of the evolutionary algorithm. This is due to the fact that changes in their specification can lead to divergent performance results. Hence in the final step, the best NN should be chosen throughout all populations with any preferences towards its generation (data selection, etc.) should be omitted.

To automate the repeating steps of population generation, mutation, selection as well as building the resulting networks, the training management software has been extended with an additional GUI, which is illustrated in Figure 5-8.



Epoch	Path
Generation1	E:\Training M
Generation2	E:\Training M

Initialisation
Max Number of:
Conv. layers: 7
Dense layers: 4
Nb of individuals: 10

Generated from: GenerationGeneration2
Restrictions
Max Number of:
Conv. layers: 7
Dense layers: 4
next populaton: 15

Pairing selection
[✓ Truncation selection (best 30)]
Fittness proportionate selection

Crossover
Number of crossings 2

Mutation
Mutation rate: 0.05

Name of network representant: Generation1
Perfomance value: 0.532

Convolutional Layers

	Active	Number of filters	Filtersize	Stride	Activation	Pooling size	Dropout
	☑	32	3	2	'relu'	2	0
	☑	16	4	1	'sigmoid'	None	0
	☐	256	3	1	'relu'	None	0.1
	☑	64	5	2	'relu'	None	0.2
▸	☐	256	3	2	'sigmoid'	2	0.3
*	☐						

Density Layers

	Active	Number of nodes	Activation	Dropout
	☑	256	'relu'	0.2
	☑	64	'sigmoid'	0
▸	☑	128	'relu'	0.1
*	☐			

Figure 5-8: GUI for evolutionary Hyperparameter optimization

Initializing population: Beginning with the first step (6) of the evolutionary algorithm, an initial population has to be randomly generated. To keep the optimization on a manageable level, the amount of HP used to characterize performance has been limited significantly. Table 5-2 shows all HPs that were taken into consideration [23, p. 100].

Table 5-2: Hyperparameters including their specifications selected for the guided tuning process

Convolution layer:

Elements	Possible specifications
Active	0/1
Filter size	2x2, 3x3, 4x4, 5x5, 7x7
Number of filters in layer	8, 16, 32, 48, 65, 128
Activation	ReLU, Sigmoid
Dropout	0, 0.1, 0.2, 0.3, 0.5
Stride	1, 2
Pooling size	None,2

Dense layer:

Elements	Possible specifications
Active	0/1
Number of nodes	65, 128, 256, 512, 1024
Dropout	0, 0.1, 0.2, 0.3, 0.5

Since convolution and dense layers are described with different properties, each list was kept separately. After the user has defined the number of convolution and dense layer as well as the total number of individuals in the initial population a function starts generating all genotypes by applying a random generator to each gene (7). As it can be seen, these genes are not necessarily a binary value, as presented in the plain algorithm in the previous section. Thus making it necessary to extend the possible selection field within a gene to an integer value ranging through all possible specification (Number of nodes in a dense layer =[0,…,4]. By knowing the number of specification type (number of nodes=4), the randomizer picks on value randomly (=2) and assigns the belonging value to the layers (=256). This is done for all genes in all layers (convolutional and dense) as well as any individual in the population. Every layer starts with a Boolean value to determine, whether the layer is actively used in the NN model. This first value enables the evolutionary algorithm to delete or revive hidden layers while giving the user the ability to restrict their maximal number.

Evaluation: In the evaluation phase, the fitness values of the existing individuals are calculated (8). In this case, the fitness function, which is used for the evaluation, usually corresponds to the optimization function. Apparently, prior to the fitness calculation, the NN has to be trained on provided standardized training data.

In this thesis testing performance of the network was chosen as the applied fitness value. For future works, a more elaborate error function can be implemented which might considers possible collisions if applying the prediction to the situation. To achieve this goal all necessary elements already have been programmed within the simulation environment. The termination condition decides whether to continue or cancel the iterative optimization algorithm. If a satisfactory solution has been achieved or a set number of iterations has been completed, the procedure ends.

Pairing selection: After determining the fitness of each individual, parents for the upcoming replacement process must be selected (9). In this project two types of selection methods were implemented, starting with truncation selection which simply chooses the individuals with the best fitness performance to reproduce. A second very common algorithm is the roulette wheel selection. Instead of just picking the best, the sum of all individual’s fitness is calculated and normed. A random number generator then selects individuals. The chance of an NN to be applied for recombination increases relative to the proportion, its own fitness value holds in comparison to the accumulated fitness values of the full population. If using this algorithm even weaker individuals have a relatively low likelihood to be selected for recombination allowing the whole optimization process to escape from a potential local minimum by adding higher diversification to the population set [68].

Recombination: The crossover can be controlled by selecting the number of crossover-points per recombination (9). A function then randomly divides both parent NNs on random locations accordingly and joins them together afterward. The structure that is applied for crossover is ordered starting with all listed convolution layers followed by their dense layers.

Mutation: A mutation rate, as a user input is required in the last step to randomly alter resulting child individuals (10). A function loops through all HPs and selects mutation objects randomly. The new specification is found by a random number generator choosing the assigned property according to the type of the mutated target.

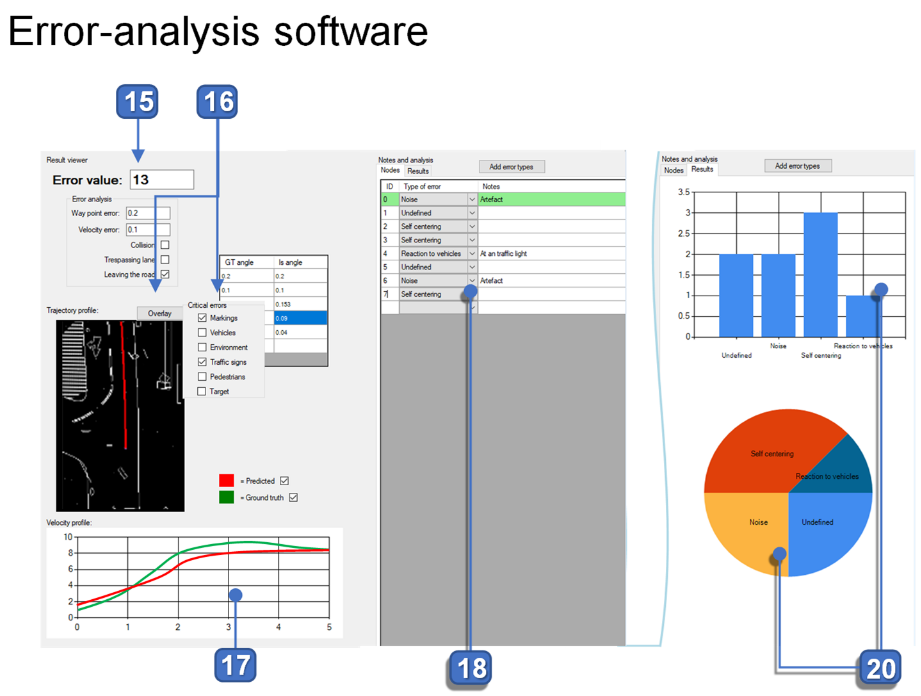
Since the automatic generation of NN Keras TensorFlow files has already been programmed and introduced in the previous section, this function has been adapted to organize and save all individuals in executable Python files that can be started without any further changes (7). This allows applying the evolutionary HP-tuning approach to run on a highly automated level leaving the user to execute the training without requiring any more profound insights into the process and current tuning state.

5.3 Error-analysis and Network performance improvements

Performance improvement of NN is highly relevant after designing and training first networks. Potential objectives for improvements can easily be found by observing and structuring errors occurring on evaluation and test data.

A common way of training and improving a network is to start training as fast as possible and then improving performance using Error-analysis. First, the network performance has been tested on the evaluation set. This includes observing a networks output and detecting common difficulties and failures. A software has been written to support this step by combining evaluation with necessary information about the Ground Truth, as well as some important vehicle states. The main GUI elements are illustrated in Figure 5-9. Since it might be crucial but difficult to recognize whether deviations from Ground Truth likely occur when a specific vehicle state is matched, this information should be visually presented to the user with little additional effort. Vehicle states in this subject include for instance reaction to traffic light changes or logic switching (15). Additionally, the deviation concerning the path and velocity profile must be illustrated to find possibilities for improvements (16, 17). Classifying errors is crucial to rank their importance by their appearance (18). A diagram, showing the number of incidents, should guide the user to potential improvements (20).

Furthermore, one has to consider the actual application of the intelligent agent the NN is implemented in. Therefore, several data points containing images which have been provided from first modules from the intelligent agent in SILAB have been added and analyzed as well.



Error value: 13

Error analysis
Way point error: 0.2
Velocity error: 0.1
Collision ☐
Trespassing lane ☐
Leaving the road ☑

GT angle	Is angle
0.2	0.2
0.1	0.1
	0.153
	0.09
	0.04

Critical errors
☑ Markings
☐ Vehicles
☐ Environment
☑ Traffic signs
☐ Pedestrians
☐ Target

ID	Type of error	Notes
0	Noise	Artefact
1	Undefined	
2	Self centering	
3	Self centering	
4	Reaction to vehicles	At an traffic light
5	Undefined	
6	Noise	Artefact
7	Self centering	

Figure 5-9: Error-analysis software GUI

In the upcoming chapters several issues and their improvement solution have been singled out and explained in greater detail. Only significant changes have been added to the list, while more basic improvements to the simulation environment remain unmentioned or have been explained in previous chapters. These mainly included constant adjustments on pedestrian and vehicle behavior.

5.3.1 Noise generation

Noise generation in this project fulfills two purposes. First is to increase the stability of network reactions by minimizing data mismatch. Since data from SILAB (in particular the line marking map) includes various artifacts, lines, and polygons that were not drawn on the initial training maps, vehicles' behavior might become unforecastable. On the other side a randomization added to the location and properties of elements increase diversity which can have a positive effect on the generalization performance of a network.

A noise generator draws randomized artifacts, including points, lines, and circles which should stabilize behavior regarding road surfaces, and architectural elements. Since most noise is expected to occur during line marking detection, mentioned geometric elements are drawn to this map type. Figure 5-10 illustrates several of the listed noise artifacts on a map.

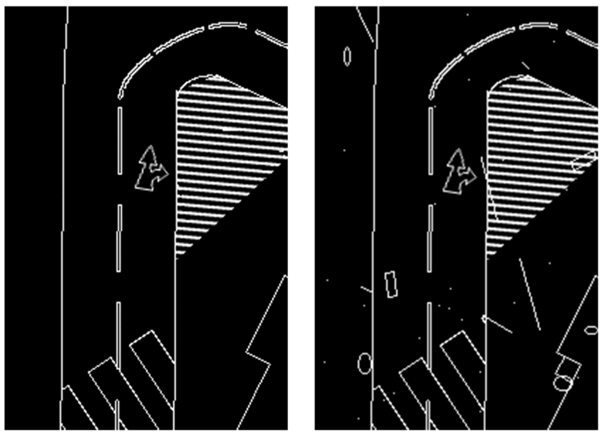


Figure 5-10: Example line markings with and without noise

A randomizer on the other hand does not add new elements to a map but changes their properties according to a predefined profile. During development, a broad range of randomization types has been added to the simulation environment including relocating of obstacles, road markings, and traffic signs, as well as the thickness and gaps of dashed lane separating road lines. A full list of all randomized objects is given in Table 5-3.

Two significant objectives are addressed by implementing randomized variation. First is to decrease efforts regarding time-consuming manual Scenario design. Adding independent variation rules for any element on the map is considered inefficient. The second reason for adding a randomizer is the to reduce simulation times. When having no randomization integrated into the software it is necessary to generate a massively larger number of Scenarios to reach the same diversity regarding the Subsimulations.

Table 5-3: Random variations and noise generated during a Simulation

Element	Noise	Range	Objective
Lines	Thickness and number of sub lines	$\{1, 2, 3\}$	mimic Kenny edge detection and different country codes.
Dashed lines	Length of gaps and thick-ness	± 10	
Road mark-ings	Orientation, location and length	Any	Add additional variance to the maps, de-creasing manual variation effort applying the Scenario designer
Hermit spines	Convexity value	± 0.3	
Obstacles	Location and size	Any	
Traffic lights and signs	location	± 5	
Environment	Fully variable street ambi-ent, buildings etc.	Any	
Marking map	Points and geometric shapes including circles squares, rhombuses.	Any	Mimic sensory noise and road surface el-ements like utility hole covers.
Target-value	Gaussian distributed noise	$\sigma = 0.2$	represents GPS/user input error and in-creases stability
Vehicles	Speed factor	$\{0.8 - 1.1\}$	Free driving speed is varying due to pref-erences of different drivers.

5.3.2 Subsimulations

An unpredictable longitudinal behavior in Scenarios involving traffic lights was observed during testing on the development test set. Until now, data is collected via a data logger, meaning that every vehicle's past state was saved. Ground Truth is saved on step one (t=1), which then is used to look backward (to t=0) through these logged data. This comes with problems when past information from the data logger cannot represent predictable future states for the vehicle. This is mainly because not all information that leads to a situation t=1 is available at t=0. An example should explain this problem which is also illustrated in Figure 5-11. If a vehicle is entering an intersection Scenario with a traffic light showing green at t=0. Two possible situa-tions may occur in the next tick. The traffic light switches to show red which results in abrupt decelerating or no changes happen, and the data grabber saves no significant differences. This leads to instability in the data set and thus influences driving behavior. It should be men-tioned that it is expected that the mentioned problem should not negatively affect driving when an extensive set of training data is addressing this situation. Unfortunately, this situation is occurring relatively rarely but having a significant impact on the weight-update process.

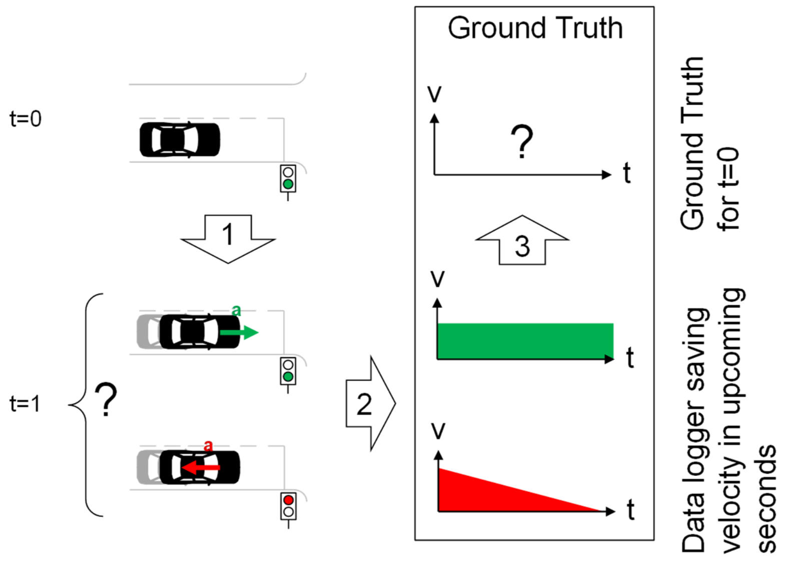


Figure 5-11: Ground truth generation while traffic light approaching without Subsimulation

To solve this problem a change in the concept of the simulation is required. Instead of merely copying simulation data into Ground Truth a secondary Subsimulation is started. It simulates any element and dynamic which the movement planning NN should learn to predict. This includes movement of and behavior of vehicles and pedestrian, while changes of traffic lights and abrupt spawning of pedestrians were omitted, as it is illustrated in Figure 5-12. After simulation, the data-grabber then can save the Ground Truth using data generated by the Subsimulation states. To initialize a behavior of natural decelerating when entering an intersection to have a chance to react to changing traffic lights, the desired velocity has been adapted within the development of Superscenarios, as it has been described in chapter 4.3.2.

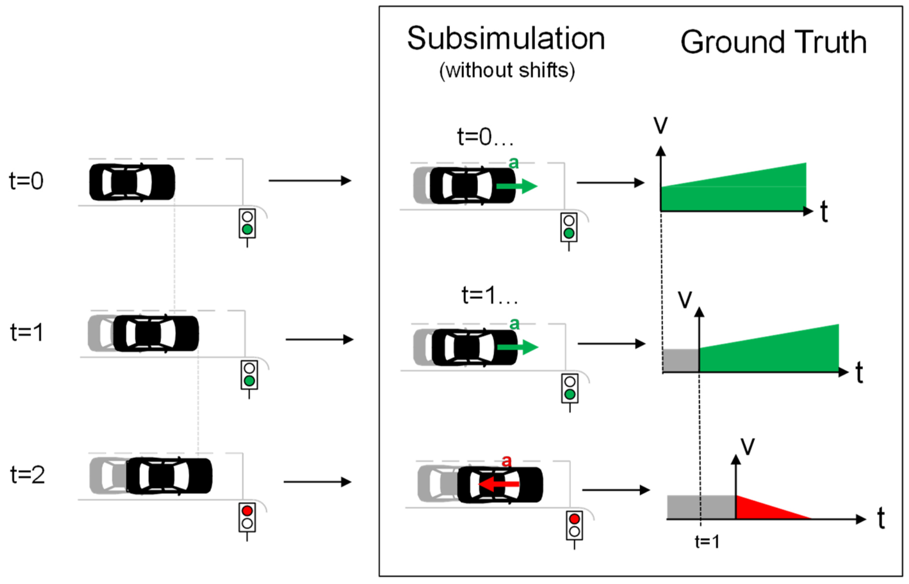


Figure 5-12: Ground truth generation while traffic light approaching with Subsimulation

5.3.3 Self-centering

When manipulating input data by adding a slight rotation or a lateral offset to it, vehicles performance instantly shrinks. This problem is expected to increase when a vehicle is actually driving with having its initial state not be continuously reseted and little errors in trajectory planning accumulate over time. No apparent reason can be given for that behavior, but it can be assumed that the generalization effect might not cover these scenarios. Training data so far only contains examples where the vehicle's starting position is located in the perfect middle and rotated towards the ideal direction of the road. When a vehicle is decentered, a situation is generated which was not covered in training data. Therefore, the network has not seen an example like this before which might result in the poor generalization effect.

Basically, the problem can be solved by adding relevant examples to the training data set. This means either augmenting existing training data or generating new ones. Last comes with additional effort, since changes have to be made to the simulation environment covered in chapter 4. Rotating images is implemented in the OpenCV library for Python. Unfortunately, images are cropped during the simulation process and therefore have no additional border which can compensate for the extra size required for offset replacement. Two extra properties were added to each vehicle state including a lateral and directional offset. Additionally, a variation step was added to the simulation that compensates these offsets during following ticks.

To keep calculation simple, old trajectory points serve as references to the offset extension. A diverging angle and offset results in a two-to-four point distance profile that guides the car back onto the optimal track. The calculation of this profile first calculates the offset that results from not correcting the vehicle's direction to the next trajectory point. This distance is then used as the maximum offset $\Delta\Psi_{max}$. This idea evolved from the concept widely known as look-ahead [69, p. 44]. A suitable Factor for offset correction is then multiplied at each trajectory point to calculate a balancing angle to guide the vehicle back to the desired track. Three types of profiles have been developed starting with a two-point correction for stiff guidance at smaller deviations to 4-point-profiles at larger offsets and curvy tracks respectively. All mentioned concepts are illustrated in Figure 5-13.

Geometrically the effect of summing up offsets and applying same 4 m distances between trajectory points is not very precise when considering larger angles. Since compensation should have the counterbalance effect already starting at smaller deviations and bring the car back on optimal track small-angle approximation for centering angles showed acceptably. Lastly, it should be mentioned that the rotation of the vehicle has affected the target-value which needed to be updated accordingly.

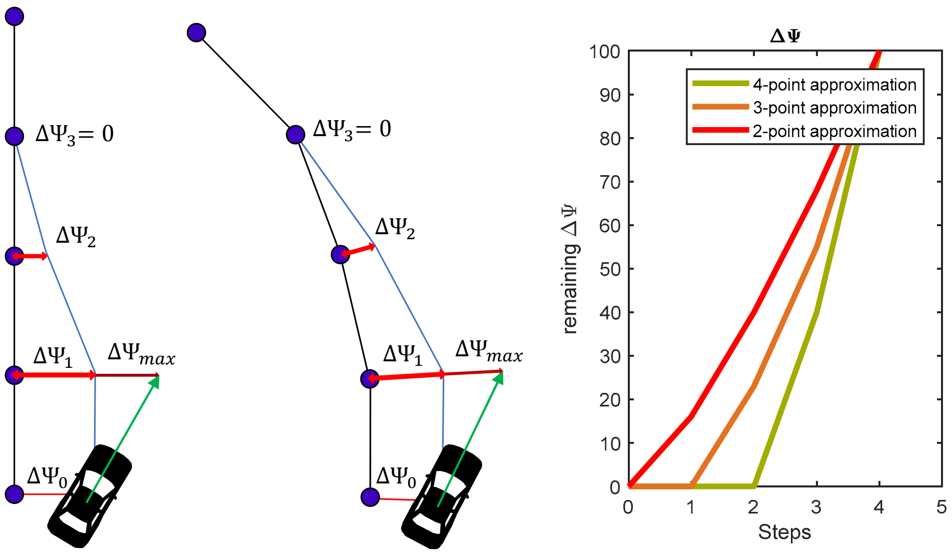


Figure 5-13: (left) three-point self-centering example (right) centering guidance profiles

5.3.4 Adjusting map resolutions

Within the 20x20 m forward-looking view moving objects were found very late, making reaction at the upper range of velocity limitations increasingly difficult. This especially holds true when vehicles are entering the visible field of the ego vehicle from a side, which often occurs in intersection Scenario without any regulating traffic lights. Considering an entering moving at 50 km/h, a reaction time of under 2 seconds is forced to find satisfactory solutions.

Therefore, it was decided to increase the potential view of the vehicle. Unfortunately, input channels of a CNN are fixed in width and length, meaning when expanding the presented range of field for vehicles, all channels need to be resized. In contrast to the size, a map's resolution (density) can be changed individually. This was used to half the resolution of the vehicle map, increasing the vehicle detection range by two. Furthermore, the target value and map were calculated on a resized area of a by 1.5 increased distance resulting in a lookahead to faster react to upcoming street turns.

The expectation, the difficulty for the network to detect vehicles including their velocity and heading angle increases was not proven. Since the vehicle map was not merely resized but recalculated by the software, current speed and heading direction were not misrepresented. The positive effects of the larger detection area, on the other hand, allowed for a better situational overview, resulting in a decreased training and testing error. The lookahead in the target value was found to have no significant impact on driving performance and has later been replaced by an unmodified value. An example for the mentioned adaption is illustrated in Figure 5-14.

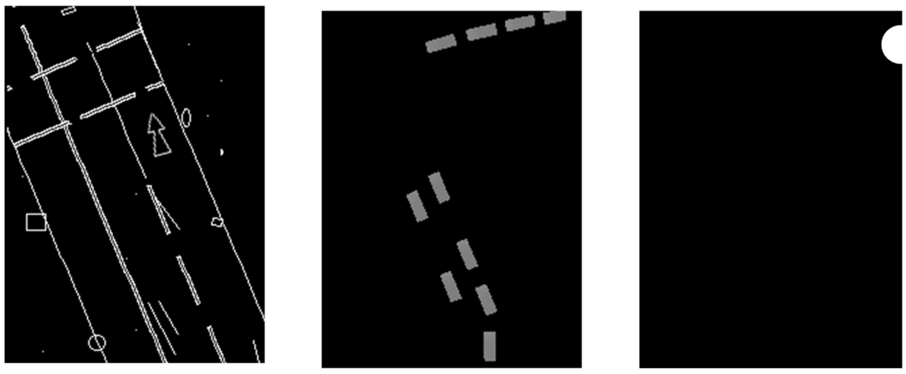


Figure 5-14: Example data point for changed resolution for vehicles and target

5.4 Learning results and Architecture

Training the Neural Networks took place on a home desktop PC containing an NVIDIA 1080 GTX, an institute server (only restricted availability including two NVIDIA Titan Xp) and the LRZ GUP-cluster (several NVIDIA DGX1 and four NVIDIA P100). During the relevant training phase, the LRZ GUP-cluster decreased its services, with only four P100 been available, because of technical difficulties on their side. This resulted in the fact that at the point of submission of the master thesis Hyperparameter (HP) tuning was still in progress. An optimization method for HP-tuning called evolutionary algorithm was programmed and integrated, which could not make use of its inherent advantages because of a lack in computational power (7 of 20 generations finished at submission data). Nevertheless, this solution for HP tuning should be described and explained so that further contributors can improve the network in the future.

Currently, the overall best performance of a trained CNN was shown by a network containing five convolution and four dense layers. The network was trained on 5,553,700 data points. These contain pedestrians (density 50) vehicles (density 150), traffic lights and signs, depending on the underlying Scenario. The HPs which result in the current best solution are listed in Table 5-4.

Table 5-4: Optimal network HP found till the end of the thesis

Hyperparameter	**Possible specifications**
Kernel Initialization	Random norm
Optimizer	Adamax $(\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999,\ \varepsilon = 10^{-8})$
Activation functions	ReLU (for all filters and nodes)
Number of layers	5 convolution and 3 dense layers
Number of filters or nodes	24, 36, 48, 64, 128
Filter size	3, 3, 3, 5, 3
Number of nodes	250, 100, 50, 10
Dropout rate	0
Error function	MSE
Total numbers of trainable parameters:	18,054,898

The validation error was steadily decreasing to a minimum MSE of 0.000019, which equals an error in 0.8° for each trajectory orientation angle and 0.21 km/h for every support point in the velocity profile. The full learning curve is illustrated in Figure 5-15.

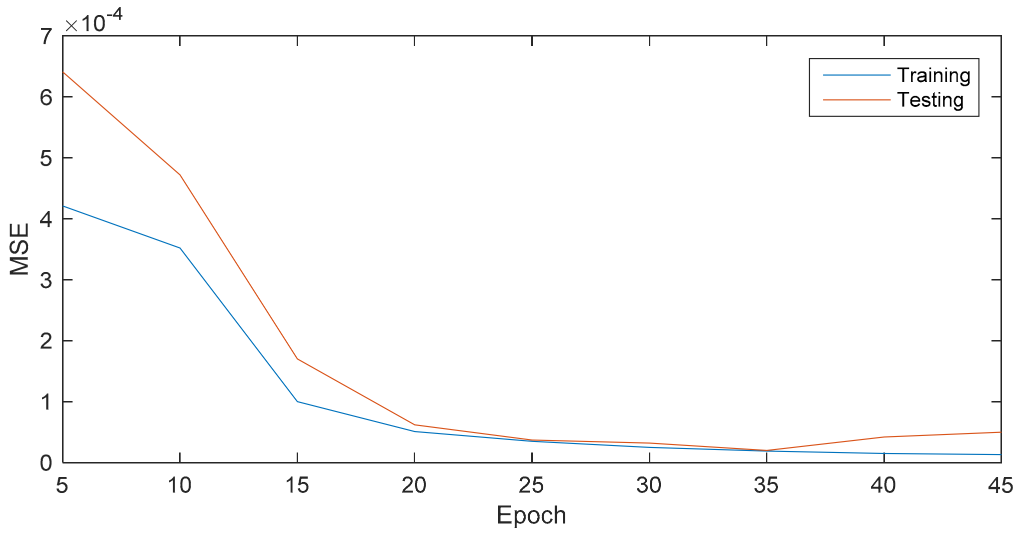


Figure 5-15: Learning curve of the best performing NN

6 Implementation and design of an intelligent Agent

After training a NN, it should be integrated into a self-developed (simulated) hardware Agent. An Agent that is acting and reacting within a complex environment to fulfill a task can be considered as an intelligent Agent. In the beginning section 6.1, some additional information to the SILAB environment is presented. Following section 6.2, the overall structure of the Agent is introduced. To gather information from the SILAB software which is necessary to calculate

Unfortunately testing and validating of the whole Agent was not possible due to time limitations and software incompatibilities on the simulation computers. However, to allow fast adaptions to the SILAB environment are described to

6.1 SILAB driving simulation environment and existing initial state

SILAB is a product of *Würzburger Institut für Verkehrswissenschaften* and is currently the applied traffic simulation environment at the Institute of Automotive Technology (FTM) this master thesis was assigned from. It differs from the self-programmed simulation environment in chapter 4 by its visualizations, driving physics, more complex road user behavior as well as providing the ability to add extending macros, called DPUs. SILAB applies a more realistic vehicle dynamics model with basic physical calculations. Additionally, humans have a sufficient behavior reacting to vehicles and elements like pedestrian crossings and traffic lights.

DPUs are written in C++ and allow extensive adjustments to the primary program. These can include simple cruise control functionality to more complex tasks like information transfer and networking. To realize a teleoperated driving experience an extension has been developed at the FTM which includes a time buffered video sender to add an adjustable delay to operator’s vision.

6.2 Agent’s structure

This section allows an overview of all developed elements necessary to develop an intelligent Agent in order to accomplish this project’s objective. The schematic including all computational modules of the Agent is illustrated in Figure 6-1. In the upcoming subsections, a more detailed view is given on various more complex elements including some additional information about SILAB, the programmed data gapper, user interfaces and vehicle control.

To start somewhere in the Agent’s computational loop, the initial vehicle state is picked, from which on the solution is explained continuously. SILAB contains all information about the ego as well as all surrounding third-party vehicles, and objects. This information is exported via an extension from an internal database and sent to the primal Python Agent via UDP. A listener waits for incoming elements and starts the primary process when a message has been re-ceived. It first transforms the coordinates from absolute locations in SILAB to required relative coordinates for upcoming operations. SILAB is using a left-hand cartesian coordinate system which has to been transformed to match NumPy requirements. Afterward, the map generator builds maps from incoming data. The map generation uses similar functions that were adapted to Python by VB.net in order not to risk disparities. Since TensorFlow uses NumPy matrices as image inputs for the CNN, objects and pedestrians are directly drawn onto them, skipping the intermediate step of drawing elements on bitmaps. This can be applied to five of the seven required inputs. Unfortunately, the target information and the road guidance map need to be provided from different sources. Since SILAB 5.0 is not capable of changing the visibility set-tings of objects throughout various visualization streams a second instance had to be started on a different PC to render an empty top-view map vision (for further explanation see section 6.3). The macro has been extended to send locations not only to the Python Agent, but also to the second SILAB instance which uses another macro to align the vehicle in the correct position. The resulting map has to be processed to meet CNN gray-scale requirements. Since taking a screenshot and applying a Kenny edge detection requires more than five milliseconds on the test system, these steps have been outsourced to the PC where the second SILAB instance is running on. Since SILAB does not have any sort of GPS built in and no navigation as required is applicable a controller supports setting of the target, an external user interface is required, which is presented in section 6.4. A buffer saves incoming target value and road marks and environment representations. When a request is received from the map generator, it pulls the last maps out of the buffer and generates the inputs required by CNN channels. The central prediction takes place in the TensorFlow backend and the trajectory with its path and velocity profile are sent back to the first SILAB instance afterward. Here the Stanley steering controller and a cruise control calculate executable commands for SILAB. Lastly, the trajectory is presented to the operator within the Field of View (FoV) which is being described in section 6.5.

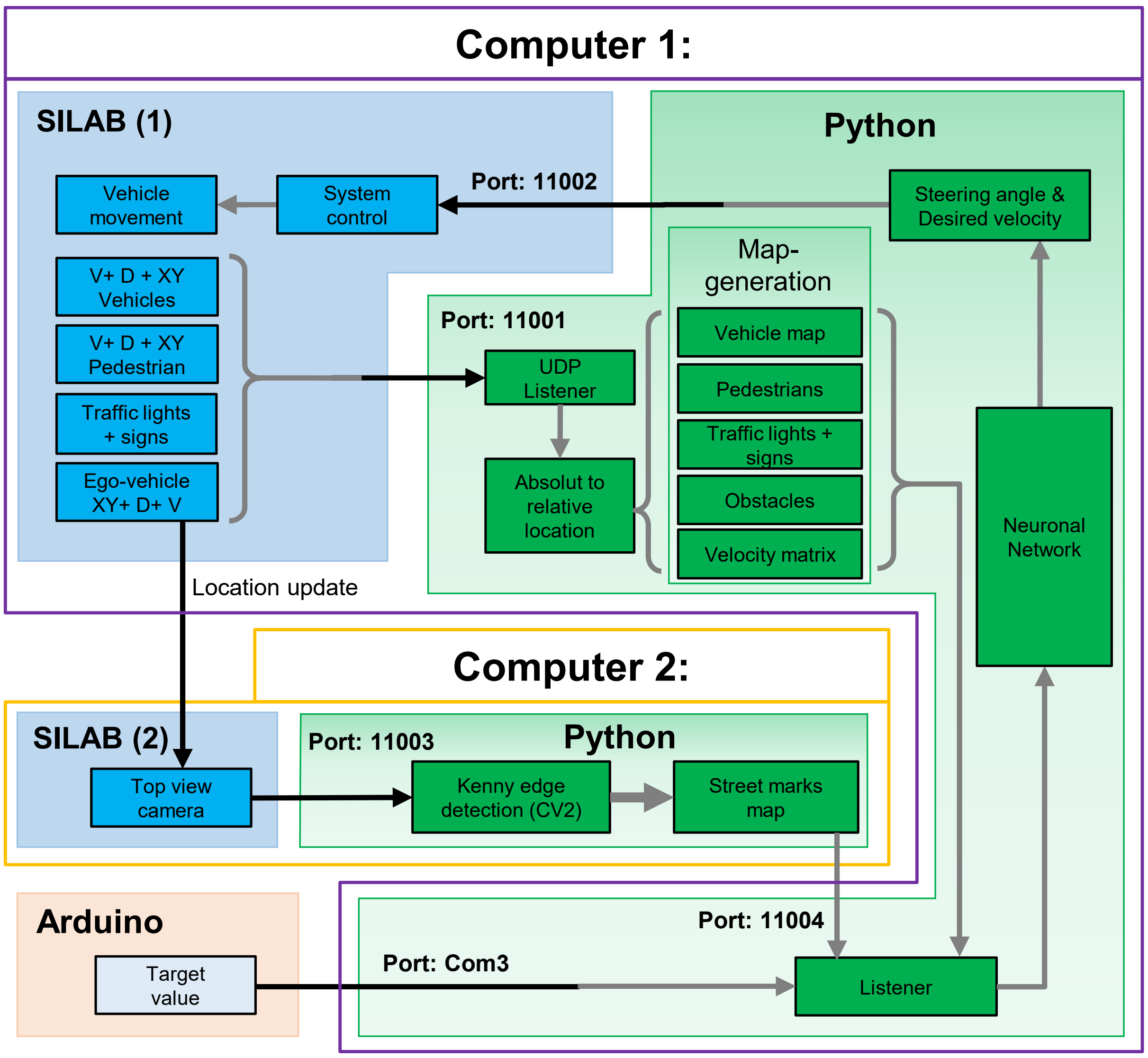


Figure 6-1: Schematics of control circuit including SILAB, Python (containing trained CNN) and Arduino microcontroller

6.3 Data-grabber and transformation

When implementing a NN into an AI solution the necessary input module is a first important crucial point to approach. In Table 6-1, all items are listed that are relevant to the network and therefore must be supplied in order to have a proper function Agent.

Table 6-1: relevant input information for the Neural Network

Map	Items and information
Vehicles	Position, heading, velocity, yaw
Pedestrians	Position, heading, velocity
Obstacles	Location, contours
Traffic signs	Location, type
Traffic lights	Location, state
Road marks and environment	Road line textures
Target	User input (see chapter 6.1.3)

To begin with, the procedure of generating environment representations for world objects like vehicles, pedestrians, static obstacles, and traffic lights must be discussed. SILAB DPUs can set up an interface connected to the internal database which is initialized by defining a polygon shaped search area around the vehicle. A query is then sent to UBD database which transmits information about any item in the mentioned polygon. Item types can be identified by an ID byte. Depending on the type relevant characteristics of objects is sorted and buffered in a related array and passed on to the Python program via UDP connection. As shown in Figure 6-1, this data then serves as a basis for the generation of related maps. Current vehicle velocity can be directly read from vehicles main status properties and transmitted as well.

Unfortunately, road marks and environment elements like lines and contours cannot be exported straight from SILAB. Even if it was possible, reassembling these elements in an image requires additional computational power. Therefore, the maps should be copied directly from SILAB. Within the options of the traffic simulation software, other cameras/visualizations can be added, defined by various properties like FoV, angle, and distance to the ego vehicle. A setting was searched that complies with the trained map properties.

As been mentioned two SILAB instances must run in parallel to generate the line guidance. This is due to the fact that it requires an empty map which does not contain any objects, vehicles and pedestrians. Upon further inquiry, the technical support stated that it is not possible to set traffic users invisible in one map while having them active on others. A workaround has been designed to tackle this problem. A second SILAB instance was started, running a map where all road users have been deleted. The DPU of instance 1 then sends heading and x,y,z coordinates characterizing the ego vehicle to instance 2 via a UDP stream. Here a Python client takes a screenshot of the map and applies a Kenny edge detector onto the image (NumPy matrix) which then transmits it to the primal Python Agent for further processing. Figure 6-2 illustrates a Kenny filter utilized for finding edges to create a road-guidance map for the Agent. An illustrating example of an applied Kenny filter on a SILAB top-view camera image is given in Figure 6-2.

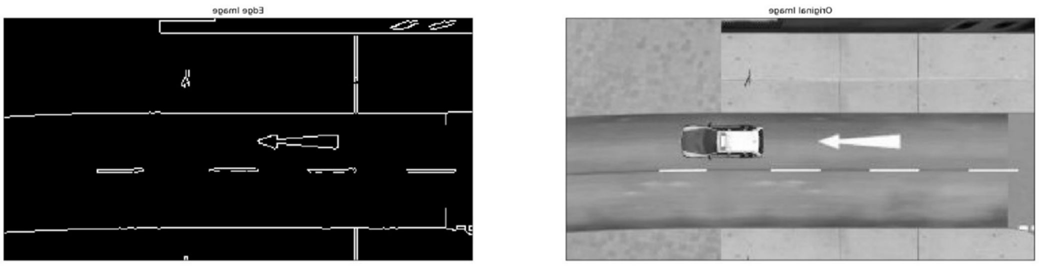


Figure 6-2: Image processing applying a Kenny edge detector

Since target information requires some additional work, the next chapter will deal with this complex.

6.4 User interface with integrated Arduino joystick

In chapter 3, it was mentioned that target information is either provided by GPS data in an autonomous driving scenario, or a User interface considering a teleoperated system. Since no GPS data are available in SILAB and movement planning algorithm is deployed in teleoperated driving sessions as well, target information needs to come from a different source. This is why a manual solution including a joystick controller has to be developed, designed and programmed in order to test the approach in SILAB.

As a hardware interface, an Arduino microcontroller has been used, which enables a fast and straightforward prototyping solution. Arduinos come with their own IDE and its easy to learn C-based API. Most three to five Volt tolerant electrical components can be connected and read by the microcontroller containing an Analog-to-digital converter with a resolution of 1024 units in the range of zero to five Volts.

In Figure 6-3 the buildup of the controller containing an Arduino Uno, an analog stick module and a slider (potentiometer). Since both elements are comparable to linear resistors a A/D converter (read-only) can transform their output voltage (adjusted by applicant inputs) to a digital signal which can be used for further calculations. For replication purposes, a schematic of the prototype is illustrated in Figure 6-3.

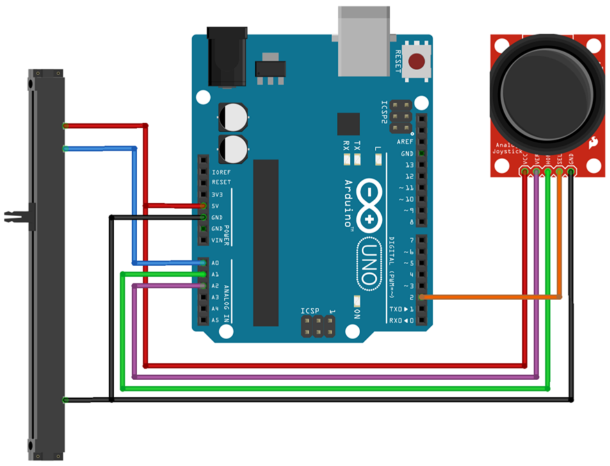


Figure 6-3: Circuit schematic of the controller with an integrated Arduino microcontroller (No LED lights have been added to ensure clarity)

The analog stick's direction is connected to the target value of the vehicle and manipulates the input of the CNN as well as indirectly changes the trajectory. Figure 6-4 illustrates the effects of the analog stick's position (orange) to the target value (red) and possible trajectory (green).

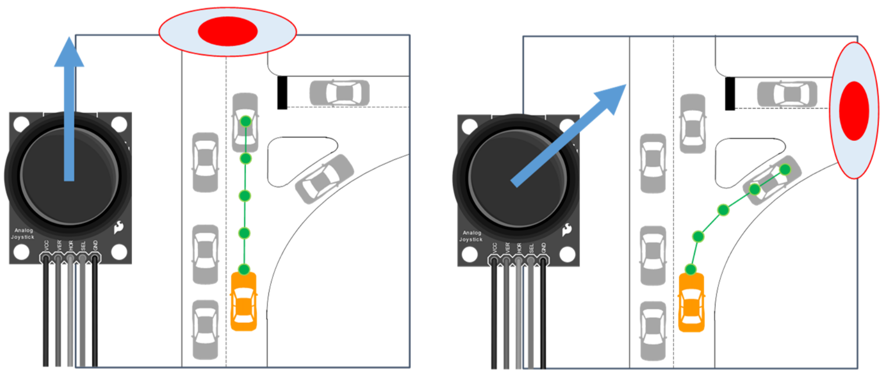


Figure 6-4: Behavior of ego-vehicles relative to analog stick direction

In addition to the analog stick, a slider was requested to adjust current velocity when needed manually. By clicking the analog stick button vehicle movement can be changed from assisted driving mode to manual free driving mode. The analog stick then adjusts its output from target providence to direct steering mode.

The software booted on the Arduino saves whether the system is in autonomous or manual mode. It then either maps the analog stick position (X and Y position from 0 to 1024) to the described target value (-0.5 to 0.5) or a steering angle. Additionally, the velocity value is normalized. The Arduino is connected to a Python code snipped via COM port and socket interpreter (115200). Every signal starts with a single digit for autonomous driving (0) and manual mode (1). Next, a float32 value transfers either the target value or steering wheel position. The location of the velocity slider completes the signal.

All three information parts are separated by a "%" symbol to allow easy access from the Python side. An example of a signal then looks like the following command, `0%0.5%100`.

To protect electronic parts from dust, moisture and unintentional unplugging of cables a cover case had to be designed. The idea of a case has been quickly replaced with the design choice of a more ergonomic operator controller. Instead of turning the analog stick towards the desired direction the user now slides a curved disk towards the desired location, which feels more convenient. The full controller contains five parts and was 3d printed at Institute of Automotive Technology as it is illustrated in Figure 6-5.

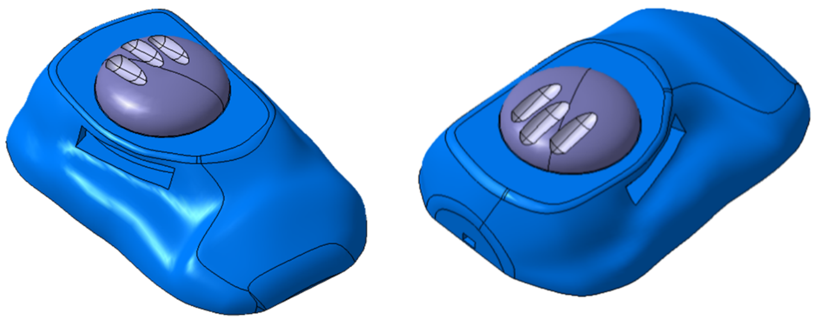


Figure 6-5: Design of an ergonomically shaped controller

While the analog stick module is already built as a self-centering electronic component the slider does not return to its initial position automatically. This was perceived as problematic during testing since the thumb can only push the slider away from the controller, with just little grip to drag it back. Therefore, a simple elastic rubber band pushes the slider against the thump and allows more comfortable controlling.

6.5 Vehicle control and trajectory visualization

As mentioned, in chapter 3.1.4 the Stanley controller has been implemented to ensure sufficient vehicle steering capabilities. The constant k was then determined by experiments. Several interpolated points were added to the ground. The value constant which resulted in the closest following at 50 km/h has been 6.2, which therefore has been chosen as parametrization.

For the driver in a teleoperated scenario, it is crucial to both see the target as well as the upcoming path points of a trajectory. For reasons of comfort and good overview which reduces the distraction, separate user interfaces were implemented in the user's field of view. The target is presented as a half transparent column painted red. It is located at the frame of the original map size (20 m to the front, 10m to the sides), as it is presented to the CNN. Points of the trajectory are represented as white disks laying on the surface of the road as it is illustrated in Figure 6-6.

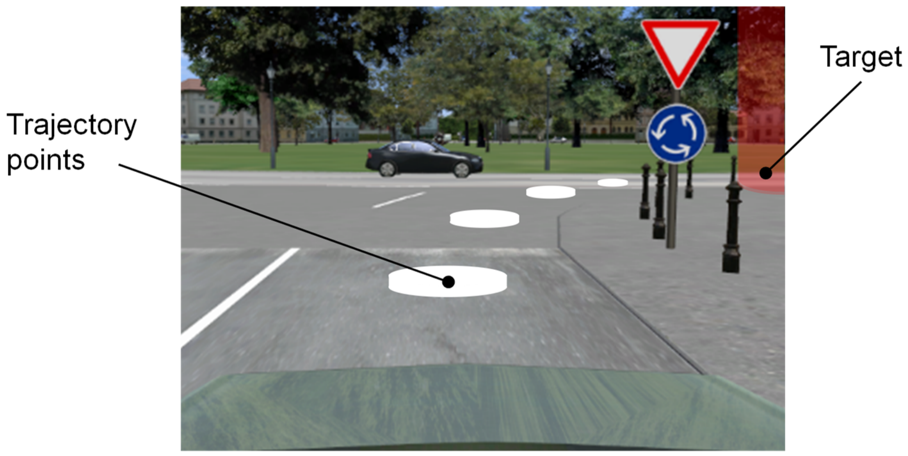


Figure 6-6: Augmented visualization interface in SILAB

6.6 Testing preparations

In this section, the testing buildup should be described, including the applied traffic situations, adaptions, and limitations concerning the SILAB traffic environment. As been mentioned, validating the agent was not possible until the submission date but will follow in a future work. Nevertheless, testing preparations should be explained to allow fast adaption in the upcoming validation process.

Starting with the testing scenarios, SILAB provides a prebuild map containing multiple inner-city traffic scenarios which represent a fictive interpretation of Würzburg. Since not all traffic situations were covered by the self-developed simulation environment representations containing three elements had to be deleted. These map parts include highway sections, ambulance clearance, as well as bypassing accidents. To enable mastering these maps, some suggestions and solutions are briefly described in the upcoming chapter 8.2, containing recommendations for future works.

In order to provide the environment representation covering road markings, which is necessary as NN input, a second SILAB instance had to be started parallelly. In this instance, any pedestrians, vehicles, and larger trees have been deleted. Additionally, to apply a canny filter for road mark detections roads that blended in, sidewalks have been shaded in darker shading for higher contrast and easier detection [70].

After training multiple NNs on desktop PCs and the institute GPU cluster as well as finishing the development of the Agent a testing phase was planned. At the beginning of this step, an error within the installed software packages on the simulation PCs was encountered. When loading the Keras models onto different simulation computers, an internal Windows pointer error occurred: `Process finished with exit code -1073741819 (0xC00000005)`. The problem arises either since different operating systems were used for the training (Windows 7, Linux) and testing (Windows 10) Hardware, or the SX Virtual Link licensing software required for starting SILAB interferes with the Deep Learning setup. The error also occurs when the necessary Windows 10 SDK had to be installed in combination with SILAB and SX Virtual Link on a Windows 7 PC.

Until the date of submission, it was not possible to fix the Keras NN model. Therefore, the Agent will be evaluated in a future work.

7 Critical reflection

From a scientific standpoint, NNs as black box models present a problematic system to work with. This especially keeps true for interpreting and predicting their behavior. Often, when relatively modern technologies are subject of scientific works, no best practices are established. This also applies to this thesis. Starting from the presented selection of input channels, shading to perception improvements and information carriage too the trajectory output, little evidence was available to prove that these concepts might succeed.

Another issue relates to the written software. Considering the approximately 60,000 lines of code written in five different languages (VB.net, C#, Python, C++, C) to develop six software prototypes, no complete polishing has been possible. Due to the occurrence of several necessary changes, the software often had to be readjusted, which reduced timeframes for polishing. However great effort on importance was attached to extendibility for further projects, troubleshooting, debugging and support features. Last includes, for instance, the Graham-Scan algorithm for finding convex hulls to create textures on road surfaces efficiently, UI debugging visualizations for traffic simulations and automatic Python file generation of NNs parametrized by an evolutionary algorithm. Furthermore, several design improvements were added due to feedback from colleagues and during consultations with the supervisor Mr. Georg.

Up to the submission date only two generations of the applied evolutionary HP optimization algorithms could have been completed. Therefore, better results concerning NN performance as part of the intelligent Agent can be expected in the future.

A last critical point concerns the missing Agent testing procedure. Since software incompatibilities occurred, no testing could have been performed with the Agent. Only its front (map generation and Arduino input) and end part (Stanly controller), as well as the necessary interfaces have been debugged and tested. As mentioned, incompatibilities between the trained Keras CNNs and the software packages on the Simulation hardware lead to errors when loading a Network model. Due to time restriction relating to the thesis as well as a tense schedule, required for designing the movement planning approach, developing the simulation environment, training a Network, and programming an intelligent Agent, just a limited time period of two and a half weeks was planned for testing. Solving these issues was not possible until the submission date. Therefore, the validation will be carried out as part of a future work.

8 Summary and future works

In this last chapter, a short summary and some continuing concepts and works regarding the presented movement planning approach are given.

8.1 Summary

Autonomous and teleoperated driving entail substantial advantages for industrial, public, and private transport. Its success is therefore in general interest. However, there are still significant obstacles to overcome before safe autonomous, or remote-controlled vehicles can be offered on the market. One major issue represents movement planning for vehicles in traffic situations including behavior prediction, planning, and execution of movement within a near field of 20 to 40-meter range. This work presented an algorithmic solution to tackle this issue using a Convolutional Neural Network approach.

Before getting into details of the movement planning algorithm, first it is necessary to delimit the approach from related topics and tasks. These projects do not include image processing and object detection, since different other student research projects search for solutions in these areas simultaneously to this thesis. Because detection and classification of objects and pedestrians around the vehicle are made available by these projects, environment representations were chosen as information mediums and inputs of the CNN. Each channel of the CNN contains information of different objects, including road markings, vehicles, pedestrians, traffic lights and signs, or static objects. This modular approach to information providence through multiple CNN channels allows further extensions in the future. As the desired prediction output for the movement plan, a trajectory has been selected which contains a geographic path and a velocity profile. For reasons of complexity reduction, a spline has been chosen as path representation defined by support points. These supports are characterized by sequentially determined angles connected by a fix distance (radius) and described within a polar coordination system. Lastly, the Stanley and a cruise controller then transfer the trajectory to actuator inputs including steering angle and velocity respectively.

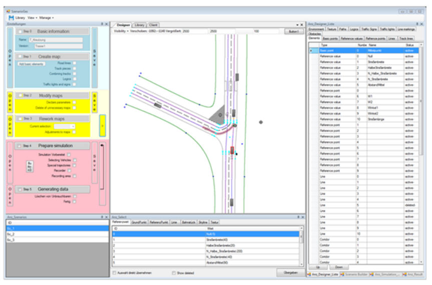
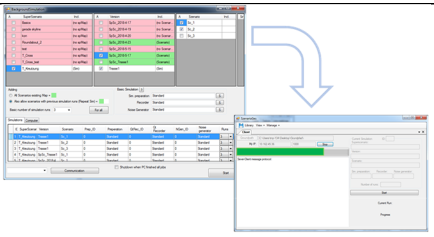
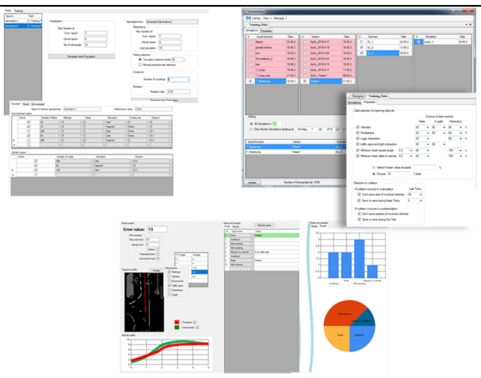
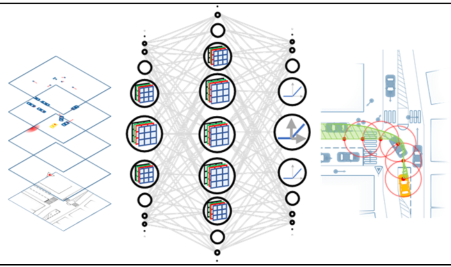
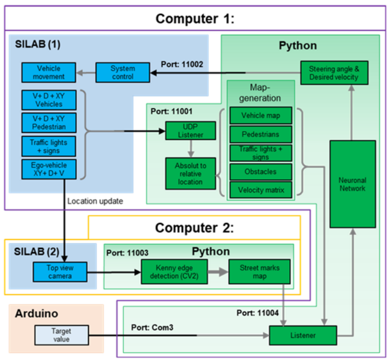
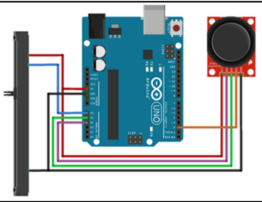
Neural Networks require a significant amount of data examples, when optimal performance is expected. This demand increases if very complex tasks must be handled, including traffic scenarios which require line detection, road following, collision avoidance and compliance with traffic regulations. Therefore, a simulation environment has been developed. It allows designing maps (Superscenarios) with various available items to choose from. Single maps will not provide enough diversity in data to expect a good generalization effect, which is why a variation interface has been added to allow fast modification to the maps (Scenarios). Vehicle data to train the model requires a traffic simulation (Subscenarios) that imitates vehicle and basic pedestrian behavior. Subsequently, vehicles' actions are recorded via a Ground Truth generator. This last step binds substantial computational resources to generate a decent amount of training examples (7 million). Since only commodity computers were available, a server-client solution has been programmed to scatter calculations amongst several computers. Lastly, a software composes environment representations in packages of 100 maps each and saves all relevant training data with their image links in a single file to allow the user to filter as well as select data that is serving the network for training and evaluation purposes.

The CNN network training was explained, and all steps have been combined to a roadmap. A profound insight into the Hyperparameter tuning approach by applying an evolutionary algorithm was introduced before the final architecture has been presented. Five convolution layers with four dense layers enabled best results comparing the errors on the evaluation set. For further improvements an Error-analysis was carried out, not only on the evaluation data but also with data from the traffic simulation software SILAB. This allowed estimating the AI behavior in a more realistic testing environment. Improvements have been implemented, including a randomizer to add noise to training data, a lateral offset extending ego vehicle's position to force self-centering, as well as Subsimulations to enhance Ground Truth quality.

Lastly, the implementation of the developed CNN into an intelligent Agent was presented. A data-grabber has been developed to collect and send data from SILAB to the movement planning application. An Arduino microcontroller integrated in an ergonomic joystick controller sends user preferred target location via a serial port interface, complementing required data for the Neural Network. All data is buffered and processed. An attached map generator transformed the data into maps that are used as input for the CNN. The trajectory is then calculated and transferred to SILAB, where a Stanley and cruise controller transforms the Neural Network output to adequate actuator inputs for operating the vehicle. A validation was not manageable in the end because of software incompatibilities and time restriction regarding the master thesis but are planned for the future.

To achieve the objectives of the master thesis seven different software programs have been developed, written in five languages (C#, VB.net, Python [Keras and TensorFlow], C, C++, and some minor MATLAB scripts) containing over 60.000 lines of code. A full list of all written software, including their purpose and some major information, is presented in Table 8-1.

Table 8-1: List of all written Software

Name	Illustration	Purpose and features
Simulation environment		**Training data generation** • Designing Superscenarios (maps) • Items including road lines, tracks, routes, environments, logics, traffic lights and signs, obstacles, and markings • Varying Scenarios (modifications) • Generating rules and proxies • Managing combinations and subcombinations • Generating Subscenarios (simulation) • Simulating vehicles' behavior (cascade and PID control) • Generating environment representations
Server-Client solution		**Distributing and managing simulation sessions** • Server • Preparing and managing simulation sessions • Distributing Jobs and communicating orders • Client • Simulating Scenarios • Generating and saving jobs on network drive (in packages)
Training management software		**Managing network training** • Providing data • Assembling, evaluating, ranking, and selecting data • Data shuffling • Evolutionary algorithm for Hyperparameter tuning • Automatize evolutionary process • Generate executable Python files • Perform Error-analysis • Presenting and visualizing training and evaluation samples • Documenting and sorting errors to guide the improvement process
Network training and evaluation		**Training and evaluating Neural Network models** • Training • Load and train NNs • Providing shuffled and batch sized data • Evaluation • Testing NNs on evaluation set for performance measuring • Processing individual SILAB examples and exporting results
Intelligent Agent	 Computer 1: SILAB (1) Python Port: 11002 Vehicle movement System control Steering angle & Desired velocity Map-generation V+D+XY Vehicles V+D+XY Pedestrian Port: 11001 UDP Listener Vehicle map Pedestrians Traffic lights + signs Traffic lights + signs Obstacles Ego-vehicle XY+D+V Absolute to relative location Velocity matrix Neuronal Network Location update Computer 2: SILAB (2) Port: 11003 Python Top view camera Kenny edge detection (CV2) Street marks map Arduino Port: 11004 Target value Port: Com3 Listener	**Implementing an intelligent Agent** • SILAB DPUs • Sending location and states of vehicles, pedestrians, traffic signs and lights, as well as obstacles • Providing Agent with a empty road guidance map • NN data input provider • Coordinate transformation • generation of environment presentations • Kenny filter for road guidance contour map • Implementing NN output to control vehicle in SILAB • Longitudinal dynamics using cruise control • Lateral dynamics applying a Stanley steering controller
Arduino controller		**Measuring and sending user input** • Measuring analog user input and convert them to digital values • Transforming signals in target value and velocity • Sending information via serial communication

8.2 Future Work

The presented approach in this work was planned, designed and implemented within the six-month limitation of the master thesis. It is important to mention that additional work is required to polish the solution, increase the range of applications, improve performance, as well as validate the approach. The following list gives guidance and some recommendations for future works from the developer's point of view.

Improving performance

Multiple resolutions and different images channels to improve forecasting: In this work all environment representations using the same size (30x20m) and resolution (1pixel ≙ 1dm). While the size is not allowed to vary amongst channels (using standard CNNs without separated branches) the resolution is freely selectable. During the designing process, the idea came up, that a second roadmap including a wider range of perception might increase stability in very complex environments by adding additional information about upcoming road sections. This especially might be considered if Lidar shadowing might decrease the range of vision and a map vendor can provide highly detailed roadmaps.

Increasing range of applications and capabilities

Adding dynamic lateral guidance: To encounter more complex scenarios with different road users that require a general deviation of static trajectories, a potential field approach that wraps tracks in their sphere of influence might be a solution. This allows improving the behavior of vehicles in scenarios where they have to deal with ambulances, passing bicycles or avoiding pedestrians by generating evasive trajectory. Vehicles which are driving on an affected track, then follow the manipulated path, as it is illustrated in Figure 8-1.

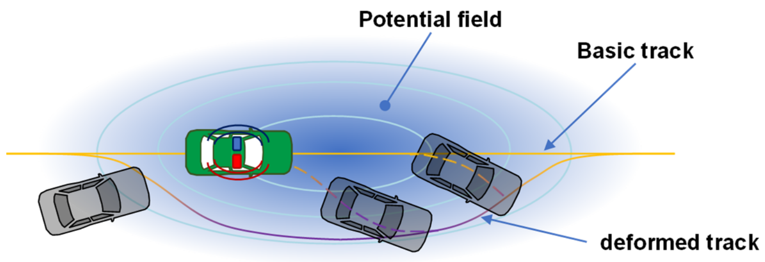


Figure 8-1: Evasive trajectory generated by a potential field

When implementing this approach to the simulation environment an additional lateral control is necessary so that vehicles retrieve the desired path if a track has been manipulated.

Adding additional environment representations as an extending information source: A project that is currently in progress at the institute contains reaction and handling of vertical irregularities like road holes on street surfaces. The channels can be easily extended by an extra map because of the modular design approach. Implementing a potential field adaption including a lateral control, (as mentioned in the previous point) into the simulation framework may add additional value to the NN's capabilities.

Developing more specified Scenarios: Currently, training data only includes typical traffic situations. Scenarios serving a more specific purpose, like overpassing or evasive maneuvers using detailed predefined paths can improve safety and situational behavior. A method to include these Scenarios in the simulation environment is to add a function that allows switching tracks or path during runtime when a particular event happens.

Implementation and validation

Validation: Since the project size was increasing over time, no detailed validation including an expert or volunteer studies could have been undertaken. In advance, the software incompatibility must be solved, occurring when the necessary software setup has been installed (chapter 6.6). The prototype is ready to use and all connections to SILAB are coded. An additional joystick is designed so that further students can follow up conducting the mentioned studies with little extra programming work to do.

Implementation of the network into a combined prototype: The main scope of this work was to design a movement planning approach. This neither included sensor perception, nor a more complex actor control for vehicle steering, acceleration, and braking. While working on the master thesis other students parallelly developed solutions for camera-based detection and classification of objects as vehicles, pedestrians, and other road users. An overarching topic for an upcoming thesis might consequently include the connection of all standalone solutions to develop a combined prototype.

List of Figures

List of Tables

Bibliography

[1] G. L. F. Falcini, "Deep Learning in Automotive Software," IEEE Software, Bd. 34, Rn. 3, p. 56–63, 2017.

[2] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-Time Motion Planning for Agile Autonomous Vehicles," Journal of Guidance, Control, and Dynamics, Bd. 25, Rn. 1, p. 116–129, 2002.

[3] T. Reitmaier, Publ., "Aktives Lernen für Klassifikationsprobleme unter der Nutzung von Strukturinformationen". Bonn: Gesellschaft für Informatik, 2015.

[4] D. Silva, "10 ways in which autonomous cars will improve our lives". [Online] available: https://medium.com/@dalton_os/10-ways-autonomous-cars-will-improve-our-lives-9870e5cbfba8. Found on: Jun. 12 2018.

[5] S. D. Pendleton et al., "Perception, Planning, Control, and Coordination for Autonomous Vehicles". Machines, Bd. 5, Rn. 1, p. 6, 2017.

[6] B. Lenz, H. Winner, J.C. Gerdes, and M. Maurer, "Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte". Springer, 2015.

[7] S. Bundesamt, “Statistisches Jahrbuch 2017”. 2017.

[8] H. Bardt, "Deutsche Autoindustrie und autonomes Fahren" Wirtschaftsdienst, Bd. 96, Rn. 10, p. 776–778, 2016.

[9] A. Scherer, "Neuronale Netze: Grundlagen und Anwendungen". Wiesbaden: Vieweg and Teubner Verlag, 1997.

[10] M. Copeland, "What’s the Difference Between Artificial Intelligence, Machine Learning" [Online] available: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/. Found on Jun.10 2018.

[11] L. Araujo and D. Santos, "Artificial intelligence and machine learning", 2018.

[12] U. Lämmel and J. Cleve, "Lehr- und Übungsbuch Künstliche Intelligenz" 2nd Ed, 2004.

[13] W. Ertel, "Grundkurs Künstliche Intelligenz". Wiesbaden: Springer Fachmedien Wiesbaden, 2016.

[14] S. Shai and B. Shai, "Understanding machine learning: From theory to algorithms". Cambridge: Cambridge University Press, 2014.

[15] K. P. Murphy, "Machine learning: A probabilistic perspective". Cambridge, Mass.: MIT Press, 2012.

[16] C. Jürgen and L. Uwe, "Data Mining". München: De Gruyter Oldenbourg, 2014.

[17] E. Alpaydin, "Introduction to machine learning", 2nd ed. Cambridge, Mass: MIT Press, 2010.

[18] R. Müller and H.J. Lenz, "Business Intelligence". Berlin, Heidelberg: Springer Vieweg, 2013.

[19] P. Louridas and C. Ebert, “Machine Learning,” IEEE Software, Bd. 33, Rn. 5, p. 110–115, 2016.

[20] M. Crusius, "Audi at NIPS: new approaches to AI on the way to autonomous driving". [Online] available: https://www.audi-mediacenter.com/en/press-releases/audi-at-nips-new-approaches-to-ai-on-the-way-to-autonomous-driving-9647. Found on May. 7 2018.

[21] J. Patterson and A. Gibson, Deep learning: "A practitioner's approach". Sebastopol, CA: O'Reilly Media, 2017.

[22] S. Haykin, "Neural networks and learning machines", 3rd ed. New York: Prentice Hall, 2009.

[23] N. Buduma and N. Locascio, "Fundamentals of deep learning: Designing next-generation machine intelligence algorithms". Sebastopol, CA: O'Reilly Media, 2017.

[24] R. Rojas, "Neural Networks: A Systematic Introduction". Berlin, Heidelberg: Springer, 1996.

[25] H. Bardt, "Autonomes Fahren - eine Herausforderung für die deutsche Autoindustrie" 2016.

[26] Verband der Automobilindustrie, "Automatisierung: Von Fahrerassistenzsystemen zum automatisierten Fahren". [Online] available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwiCpLCX6I-dAhXIZVAKHY6KByMQFjAAegQI-RAC&url=https%3A%2F%2Fwww.vda.de%2Fdam%2Fvda%2Fpublications%2F2015%2Fautoma-tisieung.pdf&usg=AO-Vaw3AJxUmk_wUXGOuUs4mfxRg. Found on May. 7 2018.

[27] P. Blythe, "Autonomous Vehicles: Some thoughts on Consumer Engagement" in Autonomous Passenger Vehicles: Institution of Engineering and Technology, 2015.

[28] E. Donges, "Aspekte der aktiven Sicherheit bei der Führung von Personenkraftwagen", Automobilindustrie, Bd. 27, Rn. 2, p. 182–190, 1982.

[29] M. Schubert, "Mathematik für Informatiker: Ausführlich erklärt mit vielen Programmbeispielen und Aufgaben", 1st ed. Wiesbaden: Vieweg and Teubner Verlag / GWV Fachverlage GmbH Wiesbaden, 2009.

[30] D. Connell and H. M. La, "Dynamic Path Planning and Replanning for Mobile Robots using RRT*", NJ: IEEE, 2017.

[31] X. Song, S. Hu, "2D Path Planning with Dubins-Path-Based A* Algorithm for a Fixed-Wing UAV", NJ: IEEE, 2017.

[32] D. González, J. Pérez and V. Milanés, "Parametric-based path generation for automated vehicles at roundabouts", Expert Systems with Applications, Bd. 71, p. 332–341, 2017.

[33] D. Kim, H. Kim and K. Huh, "Local Trajectory Planning and Control for Autonomous Vehicles Using the Adaptive Potential Field", NJ: IEEE, 2017.

[34] Y. Chen, Z. J. Yiyu, Z. Jianmin and D. Thalmann, "Accurate and Efficient Approximation of Clothoids Using Bézier Curves for Path Planning", IEEE Trans. Robot, Bd. 33, Rn. 5, p. 1242–1247, 2017.

[35] C. Liu, S. Lee, S. Varnhagen and E. Tseng, "Path Planning for Autonomous Vehicles using Model Predictive Control", NJ: IEEE, 2017.

[36] J. Ziegler et al., “Kartengestütztes automatisiertes Fahren auf der Bertha-Benz-Route von Mannheim nach Pforzheim". 9. Workshop Fahrerassitenzsysteme. Walting, Mar 2014.

[37] M. Bojarski et al., "End to End Learning for Self-Driving Cars" [Online] available: http://arxiv.org/pdf/1604.07316v1.

[38] S. Gnatzig, F. Chucholowski, T. Tang, and M. Lienkamp, "A System Design for Teleoperated Road Vehicles". [Online] available: https://mediatum.ub.tum.de/doc/1171394/file.pdf.

[39] T. Tang, F. Chucholowski, and M. Lienkamp, "Teleoperiertes Fahren: Grundlagen und Systementwurf" ATZ, Bd. 116, Rn. 2, p. 30–33, 2014.

[40] T. Tang, F. Chucholowski, and M. Lienkamp, "Teleoperiertes Fahren: Sichere und robuste Datenverbindungen" ATZ, Bd. 9, Rn. 1, p. 60–63, 2014.

[41] M. White, "Autonomous tractors on track: Case IH Magnum and New Holland NH Drive" [Online] available: http://www.aginnovators.org.au/news/autonomous-tractors-track-australia.

[42] C. Markakis and L. Barack, "High-order difference and pseudo spectral methods for discontinuous problems" [Online] available: http://arxiv.org/pdf/1406.4865v1.

[43] S. Thrun et al., "Stanley: The robot that won the DARPA Grand Challenge," Journal of Field Robotics, Bd. 23, Rn. 9, p. 661–692, 2006.
[44] I. Goodfellow, Y. Bengio, A. Courville, "Deep learning". Cambridge, Massachusetts, London, England: MIT Press, 2016.
[45] V. Sessions and M. Valtorta, "The effect of data quality on machine learning algorithms", 2018.
[46] X. Kong, H. Everett, and G. Toussaint, "The Graham scan triangulates simple polygons" Pattern Recognition Letters, Bd. 11, Rn. 11, p. 713–716, 1990.
[47] T. S. P. Riekert, "Zur Fahrmechanik des gummibereiften Kraftfahrzeuges" Ingenieur-Archive, Bd. 11, p. 210–220, 1940.
[48] Road vehicles - vehicle dynamics and rad-holding ability - vocabulary, ISO 8855, 2011.
[49] H. Winner, S. Hakuli, F. Lotz, C. Singer, "Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort" 3rd ed. Wiesbaden: Springer Vieweg, 2015.
[50] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures" [Online] available: http://arxiv.org/pdf/1206.5533v2.
[51] C. Tremblay, "Mathematics for game developers". Boston, Mass.: Thomson Course Technology PTR, 2004.
[52] Y. Simeng and O. Seiichi, "A Sequential Multitask Learning Neural Network with Metric-Based Knowledge Transfer" in 2012 11th International Conference on Machine Learning and Applications: IEEE, 2012, p. 671–674.
[53] E. S. Olivas et al., "Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques". Hershey PA: Information Science Reference, 2010.
[54] S. N. K. Pasi, "Effect of Parameter Variations on Accuracy of Convolutional Neural Network" p. 389–403, 2016.
[55] D. Mishkin, N. Sergievskiy, and J. Matas, "Systematic evaluation of CNN advances on the ImageNet" Computer Vision and Image Understanding, Bd. 161, p. 11–19, http://arxiv.org/pdf/1606.02228v2, 2017.
[56] H. Pérez-Espinosa et al., "Tuning the Parameters of a Convolutional Artificial Neural Network by Using Covering Arrays" Research in Computing Science, Bd. 121, p. 69–81, 2016.
[57] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks" in Lecture Notes in Computer Science, Computer Vision – ECCV 2014, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Hrsg, Cham: Springer International Publishing, 2014, p. 818–833.
[58] K. Wang, C. Shang, F. Yang, Y. Jiang, and D. Haung, " Automatic Hyper-parameter Tuning for Soft Sensor Modeling based on Dynamic Deep Neural Network". Piscataway, NJ: IEEE, 2017.
[59] K. K. E. Akı, T. Erkoc¸ and M. R. Eskil, "Subset Selection for Tuning of Hyper-parameters in Artificial Neural Networks", NJ: IEEE, 2017.
[60] J. Bergstra and Y. Bengio " Random Search for Hyper-Parameter Optimization" in: Journal of Machine Learning Research 13, 2012, p. 281–305.
[61] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms" [Online] available: http://arxiv.org/pdf/1206.2944v2.
[62] A. Ng, "Nuts and bolts of building AI applications using Deep Learning". [Online] available: https://media.nips.cc/Conferences/2016/Slides/6203-Slides.pdf. Found on: Jul. 03 2018.

[63] R. L. Haupt and S. E. Haupt, "Practical genetic algorithms", 2nd ed. Hoboken N.J.: John Wiley, 2004.

[64] K. Weicker, "Evolutionäre Algorithmen", 3rd ed. Wiesbaden: Springer Vieweg, 2015.

[65] M. Mitchell, "An introduction to genetic algorithms" 2001.

[66] D. Beasley, S. R. Bull and R. R. Martin, "An Overview of Genetic Algorithms" University Computing, Bd. 1993, Rn. 2, p. 58–69, 15.

[67] B. Baker, O. Gupta, N. Naik and R. Raskar, "Designing Neural Network Architectures using Reinforcement Learning" [Online] available: http://arxiv.org/pdf/1611.02167v3.

[68] T. Penchevz, K. Atanassov and A. Shannon, "Modelling of a Roulette Wheel Selection Operator in Genetic Algorithms Using Generalized Nets" Bio Automation, Bd. 13, Rn. 4, p. 257–264, 2009.

[69] J. S. Wit, "Vector pursuit path tracking for autonomous ground vehicles" Dissertation, 2000.

[70] L. Ding and A. Goshtasby, "On the Canny edge detector" Pattern Recognition, Bd. 34, Rn. 3, p. 721–725, 2001.