Freie Universität Berlin

Master thesis at Department of Mathematics and Computerscience

Intelligent Systems and Robotic Labs

Generating Data to Train a Deep Neural Network End-To-End within a Simulated Environment

Josephine Mertens
Student ID: 4518583
merte90@zedat.fu-berlin.de

First Examiner: Prof. Dr. Daniel Göhring
Second Examiner: Prof. Dr. Raul Rojas

Berlin, October 10, 2018

Abstract

Autonomous driving cars have not been a rarity for a long time. Major manufacturers such as Audi, BMW and Google have been researching successfully in this field for years. But universities such as Princeton or the FU-Berlin are also among the leaders. The main focus is on deep learning algorithms. However, these have the disadvantage that if a situation becomes more complex, enormous amounts of data are needed. In addition, the testing of safety-relevant functions is increasingly difficult. Both problems can be transferred to the virtual world. On the one hand, an infinite amount of data can be generated there and on the other hand, for example, we are independent of weather situations. This paper presents a data generator for autonomous driving that generates ideal and undesired driving behavior in a 3D environment without the need of manually generated training data. A test environment based on a round track was built using the Unreal Engine and AirSim. Then, a mathematical model for the calculation of a weighted random angle to drive alternative routes is presented. Finally, the approach was tested with the CNN of NVidia, by training a model and connect it with AirSim.

Declaration of Authorship

I hereby certify that this work has been written by none other than my person. All tools, such as reports, books, internet pages or similar, are listed in the bibliography. Quotes from other works are as such marked. The work has so far been presented in the same or similar form to no other examination committee and has not been published.

October 10, 2018

Josephine Mertens

Contents

1 Introduction

Companies, research institutes, universities and governments work together to enable cars to drive autonomously on public roads. Google's car fleet, one of the best-known examples, passed the fifth million self-driven mile in February 2018 and the number seems to grow exponentially, as shown in Figure 1. Google required many years of development and research to achieve driving on public roads. Regarding the complexity of driving tasks without human intervention, best-performing approaches use deep learning to extract patterns (see Chapter 5.2.1).

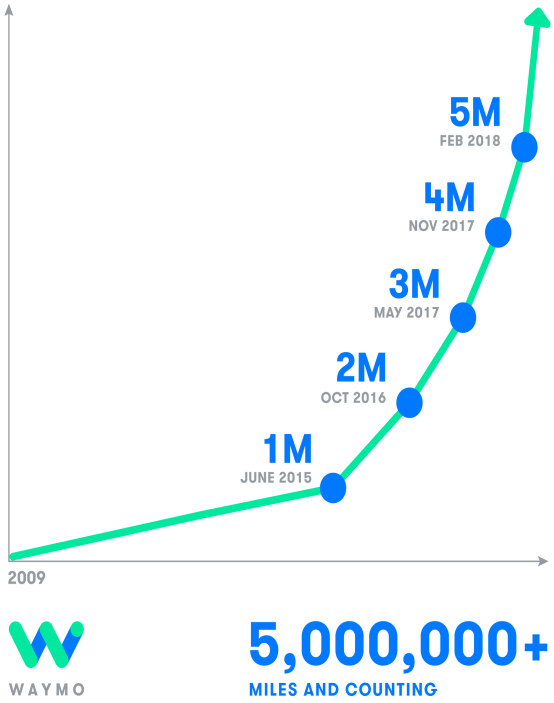


Figure 1: Total autonomous miles driven by WAYMO [27].

Deep Learning methods need a high amount of data which is difficult to get and requires significantly more time to train, compared to classical machine learning algorithms based on statistical methods such as regression. By using those methods a machine is able to generate new features by itself based on the given data. This process can be divided into two main phases. At the beginning data has to be collected, that is discussed in more detail in Chapter 2.2. After that this data will be divided into training data for the first phase and test data for the second phase.

Concerning general approaches, it is possible to drive manually around the track to record driving behaviour for the first phase. This process is very time consuming and a solution is required that generates data automatically to train autonomous driving. In this context, it shall be mentioned that there are a lot of successful groups in the field of autonomous driving. But in this work, only those who use simulated driving and published their results were considered. Looking again at Google's car fleet,

nowadays known as WAYMO [36], we see that they use simulated driving. They have driven 2.7 billion simulated miles in 2017 in combination with 5 million self-driven miles on public roads „to build the world 's most experienced driver" [27]. Additionally, the RAND cooperation determined how much miles an autonomous car needs to drive to emit a statistical degree of safety and reliability [21], see Table 1.

Table 1: RAND cooperation - Driving to Safety [21]

	Benchmark Failure Rate			
Statistical Questions	**How many miles (years*) would autonomous vehicles have to be driven...**	**(A) 1.09 fatalities per 100 million miles?**	**(B) 77 reported injuries per 100 million miles?**	**(C) 190 reported crashes per 100 million miles?**
	(1) without failure to demonstrate with 95% confidence that their failure rate is at most...	275 million miles (12.5 years)	3.9 million miles (2 months)	1.8 million miles (1 month)
	(2) to demonstrate with 95% confidence their failure rate to within 20% of the true rate of..	8.8 billion miles (400 years)	125 million miles (5.7 years)	65 million mile (3 years)
	(3) to demonstrate with 95% confidence and 80% power that their failure rate is 20 % better than the human driver failure rate of ...	11 billion miles (500 years)	161 million miles (7.3 years)	65 million miles (3 years)

*We assess the time it would take to complete the requisite miles with a fleet of 100 autonomous vehicles (larger than any known existing fleet) driving 24 hours a day, 365 days a year, at an average speed of 25 miles per hour

Looking at the facts in Table 1, a virtual driving simulation has become more necessary to fulfil such requirements. It can reduce the time and harm to people when testing a car on public streets significantly. Moreover, they enable tests in the reproduction of extreme situations, such as bad weather conditions and slippery roads. A realistic environment offers the opportunity to test new functions and refine existing ones by adapting many parameters without any risk. In this regard, this work investigates how to generate training data in automatic mode within a virtual driving simulator to demonstrate autonomous driving on a simulated circular track.

1.1 Challenges in Generating Training Data in Automatic Mode

This thesis shall to contributes to reducing the expenditure for generating data for neural networks by using a virtual driving environment. For this purpose, the following three main challenges have to be faced:

- Generating high-quality camera data that allows modelling details like rain or snow to train realistic weather conditions without the need of adding noise,
- derive or calculate trajectories for the vehicle to drive automatically with a sufficient coverage for machine learning,
- train a neural network within the simulation environment to demonstrate and evaluate the approach.

1.2 Approach

Figure 2 shows the high-level process to train a neural network to drive autonomously on a road. As mentioned in Section 1 the main challenge is given by the high amount of data that is required to train high realistic driving behaviour.

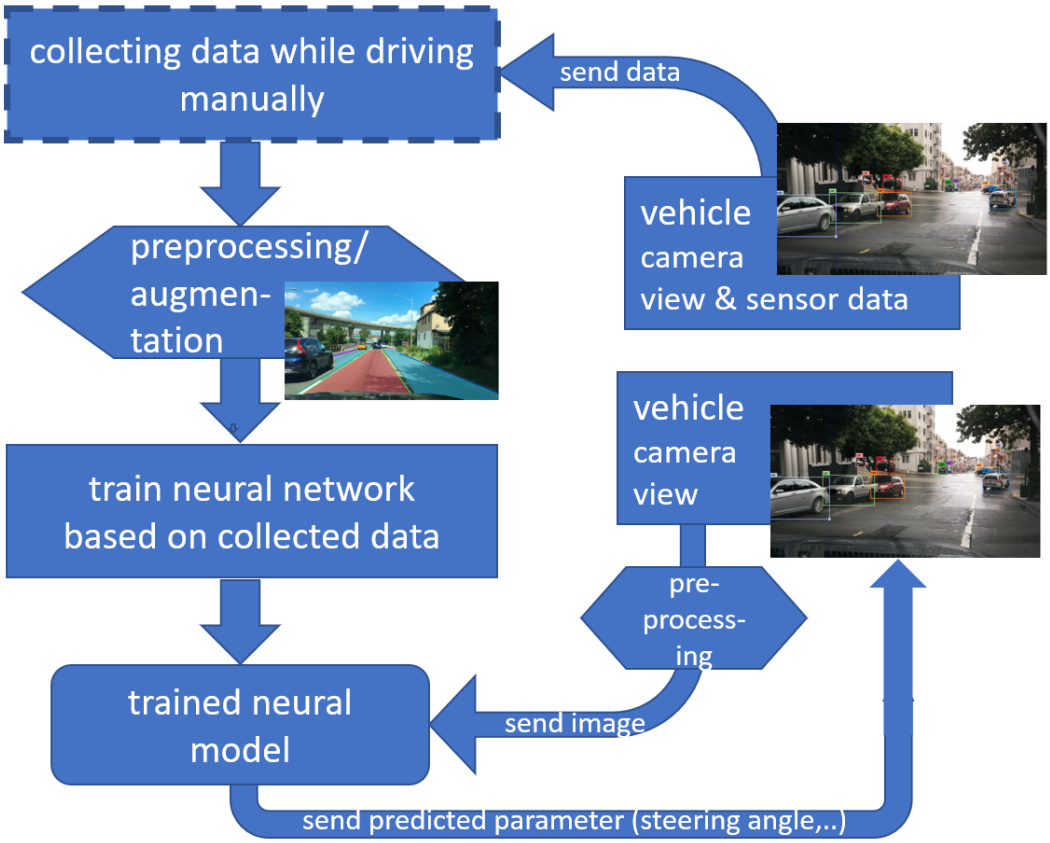


Figure 2: Process to drive autonomously on a road

To reduce the required effort when driving data is collected manually while a human driver controls the car the part of collecting data will be automated. With the training data generator, described in Chapter 2.2, it is possible to automatically generate data within a simulated 3D environment for the training and validation phase to train a neural network. The current market offers many simulation frameworks for driving tasks (see Chapter 2.3). Therefore, a simulation framework is needed which fulfil the following requirements:

- Nearly photorealistic rendering to include realistic details such as reflections and weather conditions,
- an accessible framework which can be supplemented by additional functions,
- open source and cross-platform in order to enable further development,
- an active repository which offers documentation and tutorials,

- APIs which provide access to car or extraction of camera data.

AirSim [29], developed as a researching platform for autonomous driving, meets all of these requirements. It is available as a plugin for the Unreal Engine [5]. The Unreal Engine is a complete suite of creation tools that is typically used to build highly realistic 3D-scenes for games.

Based on the test environment for the model vehicle of the FZI-Branch Office in Berlin a single-lane round track within a closed room will be built in Unreal to proof the concept. Therefore basic 3D-objects such as walls and road segments are required to build the scene which is used later to render the training and test data.

The simulation environment will be used to train a convolutional neural network end-to-end by collecting camera data from the onboard cameras of the vehicle. For this purpose, AirSim has to be supplemented by functions for generating data automatically. The data should be individually configurable within a suitable interface to fit the requirements for the respective use case. The goal is to develop an algorithm that allows following a given path. To achieve this, the target point closest to the position of the vehicle must be calculated. Once the target point is present, an algorithm is needed to calculate the required steering angle in the scene. Since undesired driving behaviour is also to be simulated, an additional method is being developed to generate specific deviations in the steering angle.

1.3 Methodology

The following work is divided into four main parts. In the beginning, common methods to collect data to train neural networks and common simulation frameworks for autonomous driving will be introduced. The appropriate 3D simulation framework is chosen by comparing the mentioned frameworks based on various criteria such as access to the code, supported operating systems or technical requirements. This is followed by a small overview of different path planning strategies in 3D environments which are suitable for autonomous driving.

The second part deals with the concept and architecture of the framework to generate data. An use case will be derived to decide how the round track will be designed and then connected to AirSim as simulation framework. Additionally, the mathematical model that is used to calculate the steering angle is described in detail. Finally, the concept of generating undesired driving behaviour is explained which is based on a weighted random value that defines the deviation from the optimal steering angle.

The third part handles the implementation of the announced approach together with a description of how the scene is built in Unreal Engine. This is followed by an explanation of the changes made in the AirSim Client to calculate the steering angle within the scene.

Following this, the implementation of a convolutional neural network (CNN) developed by NVidia [24] and an evaluation of the training results takes place. This includes a detailed description of how AirSim can be used for training and validation

of the neural network. The automatically and manually generated data will be used to train the CNN within the simulated environment. Afterwards, the results were evaluated related to the pre-defined challenges. It integrates a summary and discussion of the results as well as an outlook on its future development.

2 State of the Art

This chapter introduces the dependency between the required data to train autonomous driving and machine learning algorithms. Criteria were derived to chose an appropriate simulation environment. After that, a number of simulation frameworks that enable the exploration of deep learning algorithms are presented and compared on various criteria. This is followed by a small overview of path planning strategies in 3D environments which are suitable for autonomous driving.

2.1 Classical Machine Learning Compared to Deep Learning Algorithms

The nature of the input data needed for the neural network depends on the underlying machine learning algorithm. In general, we can roughly distinguish between classical machine learning and deep learning algorithms.

Classical machine learning algorithms require classified data as input. This means that each object in the scene is labeled. With the classified data, the vehicle can learn to distinguish between people, other vehicles, the road and objects next to the road like trees or buildings. Table 2 provides a brief insight into the suitability of some labeling types associated with the complexity measured by time and costs for the customer. Further details of the announced labeling methods are presented by the paper from Fisher et al. [38]. Looking now at large companies like Uber, Tesla, Lyft or Waymo, who have huge car fleets which collect millions of hours of driving material, we can imagine that it is nearly impossible to label all of these data.

Table 2: Brief insight about costs and complexity of some labeling types provided by the company Playment [25]

Type of Labeling	**Suitability**	**Time per Annotation**	**Cost per** Annotation
2D Bounding Boxes	Object detection and localization in images and videos	Lowest	Least Expensive
Semantic Segmentation	Pixel level scene understanding	Highest	Most Expensive
Video	Locate and track objects frame by frame in a sequence of images	Low	Least Expensive
3D Cuboids	3D perception from images and videos	High	Expensive
Polygon	Precise object shape detection and localization in images and videos	Moderate	Expensive

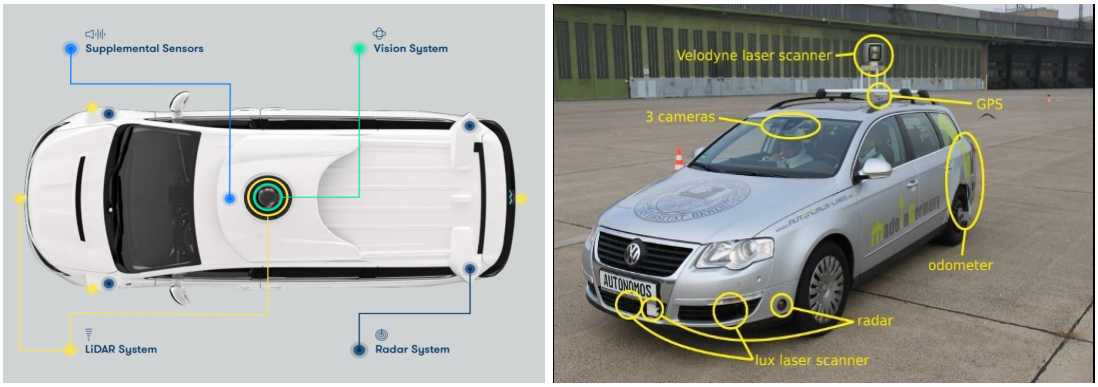
Due to the immense effort caused by classifying and labeling a scene and the fact that the labeling is usually done manually, deep learning methods can be considered as alternative. These methods only require a small training dataset and independently derive rules based on the further unprocessed input data. Often used deep learning algorithms are for example imitation [7] and reinforcement learning [10].

High-resolution images contain plenty of features. The challenge of the deep learning algorithms is to filter the right ones, for example, the recognition of road markings. Due to the high information density of the image data, deep learning algorithms require a lot of data. Furthermore, they cannot be trained in the real world or only with very high effort, because the vehicle will learn to avoid collisions with objects by driving against them. Therefore, 3D simulation environments are required to provide a safe training and learning environment with the possibility to generate as much data as the algorithm will need to learn realistic driving behaviour. In order to derive criteria that lead to the selection of the appropriate simulation environment, the following section introduces common data formats that are used to train autonomous driving.

2.2 Elevation of Data for Autonomous Driving

As mentioned in the section above cars which own the possibility to drive autonomously need at first a high amount of data for the training and validation phase depending on the machine learning algorithm. Now, the following describes the common formats and how the data for this two main phases is collected.

The type and format of the generated or collected data depend on the installed sensors on a virtual or real car and also belongs to the required information by the neural network. Normally, the system consists of high-resolution cameras, light detection and ranging sensors also known as lidar and radar systems, shown in Figure 3.



(a) Virtual car sensor system of Waymo [27]. (b) Real car sensor system of FU Berlin [2].

Figure 3: Example of installed sensors on real cars.

Together, all information gained by the sensors represents a driving behaviour related to a situation captured by one or more cameras. Training data usually consists of ideal and undesired driving behaviour, such as turning in on a straight track due to microsleep or inattention of the driver.
The data is often recorded while a human controls the vehicle through various situations over multiple hours and miles. This method was used, for example, by the researchers at the Karlsruhe KIT to create the KITTI dataset [16], by the Daimler AG R&D et. al to collect the Cityscapes dataset [9] and by Xinyu Huang et. al to collect

the Apollo dataset [20]. Further, Table 3 shows the recorded training data, collected by a team from Udacity. Where they recorded 70 minutes of driving data through mountains. Which means 223 GB of image and log data open source accessible on GitHub [34]. Each entry contains the actual position (represented as latitude and longitude), gear, break, throttle, speed and the steering angle related to an image frame, shown in Figure 4.

Table 3: Sample log of mountain driving dataset collected by Udacity

Latitude	Longitude	Gear	Brake	Throttle	Steering Angle	Speed
37.399960	-122.131840	4	0.147433	0.307836	0.005236	10.15000
37.399813	-122.132192	4	0.213535	0.149950	0.024435	0.000000
37.398688	-122.134251	4	0.147890	0.285496	0.144862	6.222222

Figure 4: Sample images from Udacity Mountain dataset [34].

The datasets include data in form of a variety of weather conditions, nature and city environments and different terrains. The simulation environment should, therefore, be able to reflect these situations and calculate different information such as the current position in GPS coordinates, the orientation of the car e.g. by an IMU sensor or a distance measurement by one or more lidar sensors. The information collected by the sensors and the captured camera views are used to set control commands such as steering, velocity or brake inputs to the car.
In the following, thus simulation environments are discussed which are used for autonomous driving research and compared according to the aforementioned criteria for selection of the appropriate simulation environment.

2.3 Simulation Frameworks for Driving Tasks and Artificial Intelligent Research

In the following, simulation environments are presented that have already been used in research to generate data for autonomous driving. Furthermore, only results from

companies and research institutes who have published their work and provide open source access to it are considered. Finally, the mentioned frameworks are compared to each other in terms of:

1. Extensibility and flexibility,
2. full access to scenery and vehicle parameter,
3. physical model,
4. sensor model integration,
5. graphical quality,
6. simulating weather conditions, different terrains.

2.3.1 TORCS

The open car racing simulator (TORCS) was first developed by Eric Espieé and Christoph Guionneau in 1999. Since 2005 it's managed by Bernhard Wymann and open source available under GNU GPL license [37] as open source software. The primary purpose is to support the development of AI-driven cars next to manual driven cars. A user can choose between popular Formula 1 car models and different racing tracks, where the variety could be extended by add-ons. A big advantage is given by the opportunity to add a computer-controlled driver or an own self-driving car to the game if it's written in C/C++. Furthermore, the user is able to let the self-driving cars drive against up to 8 other players on the TORCS racing board platform [3], shown in Figure 5. Furthermore the portability, modularity and extensibility of the framework are reasons, why TORCS is often used as a base for research purposes.

Figure 5: Sample front view torcs[3].

The Princeton Vision Group used TORCS to evaluate and test their approach to learning affordance for direct perception [5]. They collected screenshots associated with

speed values as training data, while the car was controlled manually around the track. It's also useful to capture different views of the lane and generate datasets [22].

2.3.2 VDrift

VDrift represents a car racing simulator with drift racing in mind. It's open source, cross-platform and released under GNU GPL license [35]. Created by Joe Venzon in 2005 with the goal to accurately simulate vehicle physics and to provide a platform for artists and developers. VDrift offers over 45 tracks and cars based on real-world models with several different camera modes.

Figure 6: Sample front view vdrift [35].

The BMW group developed a framework for the generation of synthetic ground truth data for driver assistance applications [17] in cooperation with the TU Munich. They modified the VDrift game engine to simulate real traffic scenarios to generate data for different image modalities, like depth image, segmentation view or optical flow. They used the integrated replay functionality and drove the track multiple times, while they record each round. In this way, a scene with multiple vehicles could be created to simulate traffic. The limited graphics resolution and low-resolution textures are the main disadvantages of this engine.

2.3.3 CARLA

CARLA is an open source framework based on the Unreal Engine [14], which is able to render high realistic textures, shadows, weather conditions and illumination, as depicted in Figure 7 [11]. It was released in 2017 under MIT license and developed by a group of five research scientists to "support development, training, and validation of autonomous urban driving systems" [12].

Figure 7: Scene with vehicle parameters captured from the town environment in CARLA.

As part of the research, it's the only framework, which provides an autopilot module. This is realized by encoding driving instructions to the ground, see Figure 8. A function iterates over each pixel of the scene and calculates the direction to encode the associated colour. The green colour defines the driving direction for the lane. White colour points signal intersection areas, where the car can decide between multiple routes. Areas next to the road are displayed with black pixels. The car does not get an instruction and the car will turn left to get back to the road.

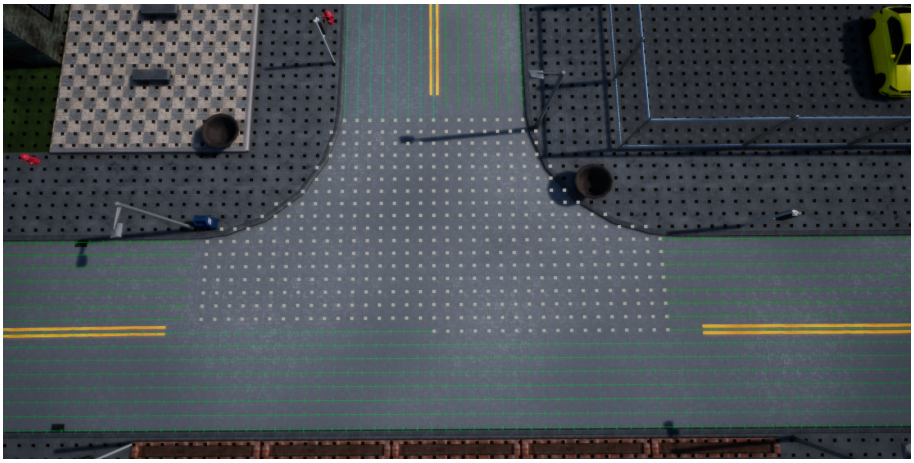


Figure 8: Debug view of road instructions rendered in the scene [12].

2.3.4 Airsim

AirSim is developed by the Microsoft Research Group with the aim to provide a platform to experiment with deep learning, computer vision and reinforcement learning for autonomous vehicles [30]. To use this framework an 3D environment built with the Unreal Engine is needed first. For this, there are prebuilt binaries of some envi-

Figure 9: Screenshot of the Neighborhood Package [13]

ronments such as the neighbourhood environment shown in Figure 9. These binaries can be used to train neural networks or t collect data.

Even though the age of just one year, there are many research institutes, which are using AirSim in their projects, who are listed on the GitHub repository [1]. The centre for intelligent systems laboratory, CS, Technion is one of them, who use AirSim in a self-driving simulation manner for an autonomous Formula Technion project [19]. Their approach was to teach a formula car to drive fully autonomously through the racing track, based on reinforcement and supervised imitation learning approaches.

This work only considers open source frameworks. Therefore all frameworks offer the possibility to access and extend the entire structure and parameters of the vehicle. Furthermore, all frameworks have a physics engine that can represent realistic driving behaviour. Related to section 2.2 radar such as Lidar, IMU and GPS sensors are often used in reality. The whole sensor suite is only available at CARLA. AirSim provides the mentioned sensors except for the Lidar sensor. Simulation frameworks such as VRep [8] or Gazebosim [15] offer an even larger sensor suite but are not considered due to the low frame rate during the scene rendering. Furthermore, it is not possible to build a photorealistic 3D environment with different weather conditions. But this is possible by using the Unreal Engine which comes with an editor to build realistic environments.
Therefore only AirSim and CARLA are considered in the following. Looking at CARLA there is only a small number of tutorials and documentation available. Furthermore, it is written for Linux operating system and difficult to install on other operating systems. Therefore the Unreal Engine in combination with AirSim will be used as an appropriate simulation environment. The advantages of AirSim are discussed in detail in Chapter 3.3.

2.4 Methods to Follow a Path in a 3D Simulation Environment

If the vehicle shall be able to drive autonomously in the 3D environment a path to follow is required. There are different methods available to achieve this. Codevilla et al. developed an algorithm based on conditional imitation learning that allows an autonomous vehicle trained end-to-end to be directed by high-level commands [7]. The driving command is then produced by a neural network architecture. A further approach is given by Chou and Tasy [6], who developed an algorithm for trajectory planning which uses a range sensor to take the path where no obstacle is detected. A third method developed by Chue et al. uses predefined waypoints, which are the base frame of the curvilinear coordinate system. The optimal path is derived by considering the path safety cost, path smoothness and consistency.

The calculation of trajectories is a complex task, with a high computational effort. Therefore the approach of Chu et al. is adopted. Based on the geometry model of the road, a spline is calculated from several points to calculate the steering angle to the nearest point on the path. To enable driving on alternative roads, the steering angle is adjusted using a distribution function and a given error rate.

3 Concept and Architecture

This chapter presents an approach for generating training and test data to train a neural network on how to drive autonomously in a simulated environment. This includes the experimental setup together with the chosen framework AirSim and the mathematical model, which calculates the steering angle within a geometrical model. At the end, I introduce an algorithm to generate ideal and undesired data by a given distribution function and an error rate.

3.1 Approach for Generating Training and Test Data

The high amount of required data to train realistic driving behaviour is the main bottleneck behind the concept of autonomous driving, as mentioned in Chapter 2.1. Therefore a data generator, shown in Figure 10, should be developed.

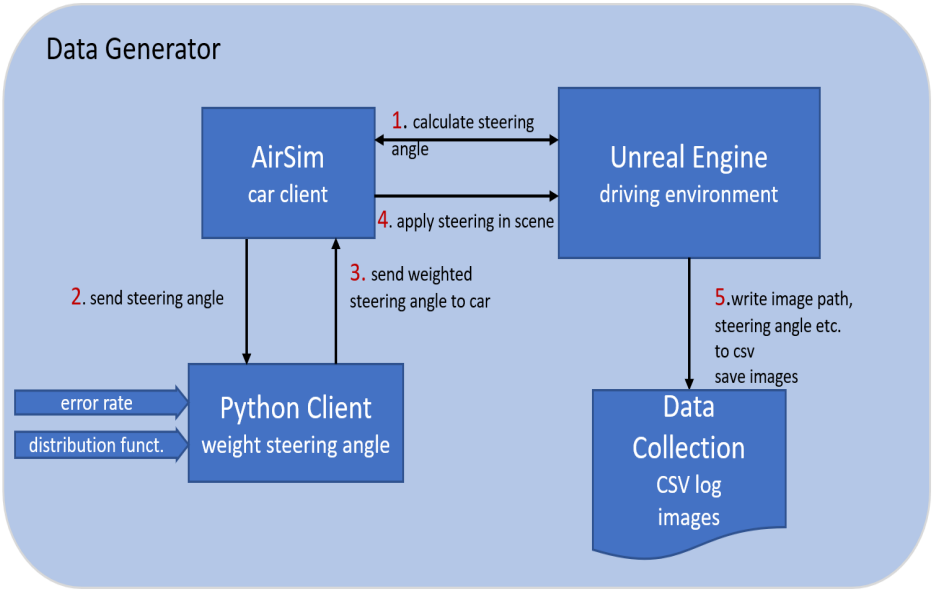


Figure 10: Process to generate data within the driving environment

The approach, shown in Figure 10, is to use AirSim to calculate a steering angle within the driving environment (step 1). After that, the steering angle is passed to a python client (step 2) which calculates a randomized weighted offset that is added to the steering angle. The calculated offset should depend on an input error rate and on a given distribution function. The resulting angle can then lead to the ideal or undesired driving behaviour. In step 3 and 4, the weighted angle is applied to the car within the driving environment by AirSim. The last step handles the writing of image files and related information such as the applied steering angle, the image path, the position data of the car and the distance between the car and the centre line of the track.

With the aim to generate realistic driving behaviour it is important to simulate the ideal driving behaviour in combination with undesired one. Undesired driving behaviour describes situations where the driver steers the vehicle in a direction that does not correspond to the destination for example, due to carelessness or fatigue. Furthermore, the steering angle of human drivers varies slightly while driving, for example, to react to road irritations or to avoid holes or small objects. These examples can be simulated by a deviation of the calculated ideal steering angle. This deviation is calculated using a distribution function that prefers small deviations and decreases the probability of larger deviations. The ideal driving behaviour means that the car is driving as close as possible to the centre line of the single-lane track. Thereby the quality of the calculated steering angle is measured by an offset between the middle point on the front axle of the car and the nearest point on the centre line. An increasing offset will be interpreted as undesired driving behaviour. Furthermore, the position of the car will be reset if the car leaves the track which results in a deviation that exceeds the width of the road.

To proof, the presented approach the following section specifies a use case for autonomous driving on a simulated 3D round track. The basic constraints and requirements in combination with a detailed overview of the reasons to choose AirSim and the Unreal Engine 4 takes place.

3.2 Challenges to Accomplish

The driving environment consists of a single-lane roundtrack with multiple turns of different degrees. At first, the vehicle should learn to handle turns while the velocity remains unchanged.

Within this testing environment the following challenges need to be accomplished:

- Computation of optimal path for the round track,
- generate different driving profiles to simulate variance in driving behaviour and to generate different viewing angles to the scene,
- calculation of an optimal steering angle, which leads the vehicle to the middle of the road,
- generation of undesired driving behaviour according to a given proportion,
- generate 2D images with labeled data.

For this a driving environment for simulating a car on a round track and a an expandable 3D framework that allows to access the scene geometry and extending the API by a mathematical model. Both Requirements are met by AirSim and Unreal Engine, that are explained in the following section.

3.3 Simulation Environment

As mentioned in Chapter 1.2 Unreal 4.18 will is used as game engine in combination with the framework AirSim. AirSim met all the requirements mentioned in Chapter 2.3. It is completely open source with full access to the code, supports the common 3D-Model formats and provides many functionalities for autonomous research. In the following section, AirSim is explained in more detail.

AirSim allows users to experiment with deep learning algorithms for autonomous vehicles. The framework was developed by the Microsoft Research Group from Redmond, the USA with the aim to provide an artificial intelligence research platform for everyone. In the beginning, only multirotor vehicles were covered and in October 2017 the developers updated the simulation of a car model. There are regular updates with the help of the community to offer the same functionality as for the drone model.

The main benefits of using AirSim can be separated in four categories:

- **Usability:** A Framework designed to be used by professionals and beginners. There are many tutorials, documentation and support from the community. The provided binaries of different environments such as the neighbourhood environment enable a good entry for testing the vehicle control system through an API.
- **Performance** The Unreal Engine offers a high computational performance, by the fact that it is written in memory-optimized C++ code, which results in a more efficient use of the CPU and GPU. Furthermore the engine provides photo-realistic rendering in real time.
- **Programmatic flexibility:** Developed as a plugin and based on the Unreal Engine 4, AirSim can be used in every Unreal project. The vehicle can be controlled by external APIs with support for multiple languages like C++, Python, Java or C# and communicate through RPC messages. These APIs are also available as a part of a separate independent cross-platform library [30].
- **Expandability:** The API can be extended by adding additional functions on server and client base. In this way, many calculations related to geometric parameters can be published and interpreted on python site.

In addition to the above-mentioned advantages, the following disadvantages must also be taken into account.

First of all, many of the functionalities for the drone model are not yet available on the car model. Therefore functionalities like driving to a position are missed and needs to be implemented for the car model. Furthermore the sensor suite only contains barometer, GPS, distance, magnetometer and an inertial measurement unit (IMU). But a lidar sensor to recognize objects next to the vehicle is missed. One more disadvantage is given by a decreasing framerate when using more than four cameras are to render the scene views.

The high flexibility of AirSim allows the development of modules in different lan-

guages. Thus, for example, simple calculations for image processing that are necessary for training neural networks can be done with Python. Furthermore, the communication via RPC messages offers the possibility to control model vehicles directly and to connect them to the simulation environment. It is possible to create almost photorealistic scenes with the Unreal Engine, which can depict real application cases. The sensors suite is constantly being expanded, so that obstacle avoidance with lidar sensors will be available soon. The rapid development and the high number tutorials together with an excellent documentation finally led to the decision to use AirSim. The mentioned disadvantages are not relevant for this work, because neither more than three cameras are used nor a collision avoidance is planned.

3.4 Calculation of a Steering Angle in a Simulated 3D Environment

Regarding the first step from Figure 10 a mathematical model is required to calculate a steering angle to reach a target point on a path. Thereby it is supposed that the targetpoint lies inside the turning radius of the vehicle. Furthermore the physics model of the car is required, to get the position of the tires, the offset between front and rear axle L and the wheelbase T, shown in Figure 11. Looking at real cars, we find complex steering mechanisms with different mechanics and linkage systems, depending on the vehicle manufacturer. The book "The Automotive Chassis" [26] from Reimpell et. al provides a summary of the common steering systems. In general, the four wheeled vehicles uses four-bar linkages, which are often based on the "Ackermann Steering Model" [39]. In the following the Ackermann model is briefly explained. It is used as basis model and has been modified with regard to the usecase, described in the last section.

Ackermann Steering Model

The concept of the Ackermann Steering calculates the steering angle for the inner and outer front tires. This means that angle of the outer tire δ_o and the inner tire δ_i depends on the wheelbase T, shown in Figure 11. An imaginary tire in the middle of the front A and rear B axle is assumed. Then A will be rotated by a given δ to calculate the perpendicular bisector of A. Now, the radius r represents the distance between B and the intersection point C.

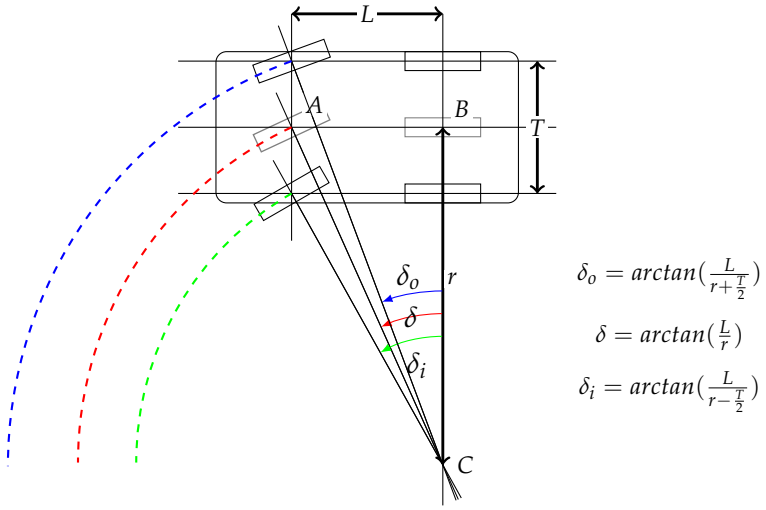


Figure 11: Concept of Ackermann Steering

Steering Model Based on a Given Path

The Ackermann steering model applies a given steering angle to the tires. In this work the steering angle and therefore the rotation of the inner tire and the intersection point C are unknown and needs to be calculated as C' by a target point such as $Q1$ or $Q2$ on a given path, see Figure 12.

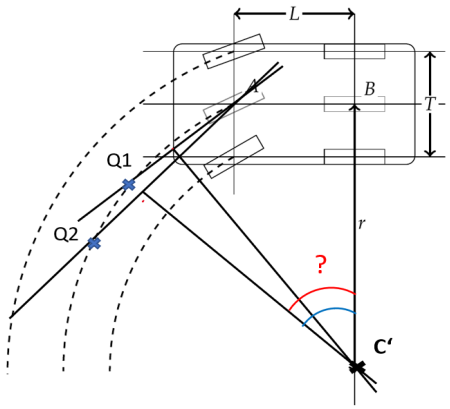


Figure 12: Difference to the Ackermann model.

The Ackermann Model is adjusted by the calculation of the perpendicular bisector between the target point Q on the path and the middle point A on the front axle. According to this step, the intersection point on the rear axle needs to be calculated in an additional step. In the following, the calculation steps of the mathematical model, shown in Figure 13 are listed:

(i) Calculation of the middle point of the front axle $A := F_L + \frac{1}{2}(F_L + F_R)$.

(ii) Calculation of the nearest point Q on the spline *path* related to A, by using a spline library function from Unreal.

(iii) Calculation of the perpendicular bisector $p(x)$ of A and Q.

(iv) Calculation of the line $r(x)$ through the backwheels B_L and B_R.

(v) Calculation of the intersection points
C of $p(x) = r(x)$
D of $j(x) = r(x)$.

(vi) Calculation of the steering angle $\measuredangle HCD := \sin(\alpha) = \frac{\overline{HD}}{\overline{CD}}$.

(vii) Calculation of the relative position of the target point Q and the vehicle position at A to get the steering direction by extending the 2D points A, B and Q by a third coordinate with z = 0. Then the calculation the crossproduct is performed to examine the value of the resulting z-coordinate to deviate the relative position: z = crossproduct(A-B, Q-A)

$$direction = \begin{cases} \text{go ahead,} & \text{if } z == 0 \quad (1) \\ \text{turn left,} & \text{if } z < 0 \quad (2) \\ \text{turn right,} & \text{else } z \geq 0 \quad (3) \end{cases}$$

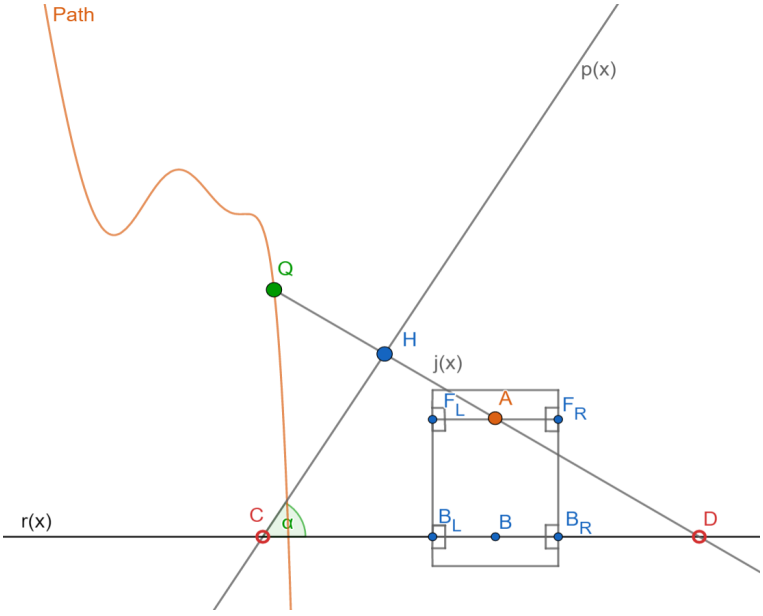


Figure 13: Model to calculate steering angle for a given target point on path

The below special cases are also handled by this model:

- If the target point lies in front of the vehicle p(x) = horizontal and j(x) = vertical, then the algorithm stops and returns 0 for the steering
- If the linear function r(x) or j(x) of the type $f(x) = mx + b$ has vertical or horizontal direction, they are defined on the constant parameter x = b or y = b

Finally the presented model calculates a steering angle that leads the car to drive on the centre line if the starting position of the car is near or on the path of the single-lane track. The steering angle will be passed to the python client and used to calculate the randomized weighted steering angle to generate ideal and undesired driving behaviour which is described in the following section.

3.5 Distribution Model for Deviations in Ideal Driving Behaviour

As per the challenges from Section 3.2 the vehicle should have the opportunity to drive different routes along the track with an adjustable proportion of undesired driving behaviour. Furthermore the decisions of the algorithm should be weighted by the use of driving profiles. This profiles can be simulated with the use of a specified distribution function and an error rate. In the following, the explanation of the three main steps takes place.

Step 1: Binary Decision

In the first step the algorithm decides about the generation of an ideal or undesired steering angle. The decision is influenced by a given error rate $e \in [0, 1]$:

1. Pick a uniformly distributed random value p within the interval $[0, 1]$

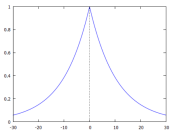
2. Interprete steering behaviour by p and e:

$$\text{type of angle} = \begin{cases} \text{ideal angle}, & \text{if } \; p > e \quad (4) \\ \text{undesired angle}, & \text{else } \; p \leq e \quad (5) \end{cases}$$

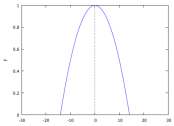
If ideal steering behaviour is chosen, the algorithm stops and applies the steering angle to the vehicle. Otherwise the algorithms goes to the next step.

Step 2: Generate Undesired Data by a Distribution Function

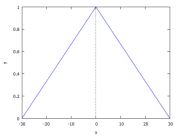
Related to the fact, that the generation of ideal driving situations is preferred the steering angles have to be weighted. The weighting factor is modeled by a distribution function. Figure 14 shows three different weighting functions, consisting of an exponential(EXP), inverse square(INV) and a linear function(LIN), where $\hat{x}$is defined bei the calculated steering angle from the scene.



(a) EXP $1.1^{\pm x}$



(b) INV $-0.005\,x^2 + 1$



(c) LIN $\pm\frac{1}{30}x + 1$

Figure 14: Selected distribution functions to weight steering angles

EXP The exponential distribution prefers ideal driving behaviour in the lower range. Undesired behaviour is possible, but the weighted steering angle is often near the ideal angle. Larger errors are more likely with a high the error rate. Therefore these type of function is good to simulate small steering movements for example when the driver is unfocused.
These kind of functions are defined by $f(x) = a^{\pm x}$.

INV The inverse square distribution function estimates almost correct driving behaviour with small errors. Larger errors are only possible if the intersection points on the x-axle are the minimal and maximal steering angle.
These functions are defined by $f(x) = -a\,x^2 + b, a \geq 0, b \geq 0$.

LIN The linear distribution weaks the error probability with an increasing angle. The probability of larger errors which can represents critical situations such as microsleep can be simulated if the intersections on the x-axle are greater or equal the minimum or maximum steering angle.
These functions are defined by $f_(x) = \pm a\,x + b, b \geq 0$.

The presented functions from Figure 14 are determined by using Maxima [32]. The aim is to simulate realistic behaviour, therefore a mathematical function is needed which preferr a desired value. This is given by normalizing the range of possible steering values. These examples influences the rsulting steering angle in different ways therefore it is important to define the distribution function $f : [-45, 45] \rightarrow [0, 1]$ as input parameter for the algorithm. $f(x)$ calculates possible deviations $n \in \mathbb{N}^+$ from the optimal steering angle within the given range. This range is divided in n sections of an equal size and we calculate the arithmetic mean. Example: $n = 4$ in the range $[-30, 30]$

1. (-30, -15), mean: $\frac{-30-15}{2} = -22,5$.
2. (-15, 0), mean: $-7,5$.
3. (0, 15), mean: $7,5$.
4. (15, 30), mean: $22,5$.

In the next step we calculate $f(x_i) = y_i$ for each x_i of the means. Then we throw the dice n-times, to get the values p_i and calculate each $y_i\, p_i = E_i$. Finalizing all x_i were sorted by E_i and we pick the X_i with maximum E_i.

Step 3: Filtering of Invalid Steering Angles

The last step takes care of a valid steering angle in the given range (turning radius). The result of the second step calculates a deviation from the optimum $\alpha_e = \alpha + x_i$, where α describes the optimum steering angle and x_i the calculated deviation. Therefore it's possible to get a value that's out of the range. This case is handled by calculating an α'_e, which is represented by the maximum of the minimum from maximum angle and the offset $\alpha'_e = \max(-30, \min(30, \alpha_e))$.

To summarize, the simulation environment consisting of the Unreal engine and the simulation framework AirSim is used to represent the use case fo generating data to train autonomous driving. By the fact that AirSim does not have a function to drive along a path an own approach is developed to calculate a steering angle to a given target point. In order to avoid overtraining of the neural network and to drive alternative paths, a method was presented to calculate a weighted steering angle that can diverge to the ideal steering angle. The following chapter summarize the process to implement this approach.

4 Generation of Training and Test Data with AirSim

This chapter deals with the implementation and evaluation of the concept presented in Chapter 3. The calculation of the steering angle requires the possibilitie to add B-Spline objects to the scene. This is not given by the prebuild test environments, that are available in the GitHub repository. Therefore the creation of a separate test environment is required. The first section 4.1 describes the generation of the simulated environment with Blender 2.79b and the Unreal Engine 4. After that, the software architecture of AirSim is explained to get a better understanding how and why the API has been extended. Finally it is shown how the data generator can be used with the simulation environment and different parameter.

4.1 Modelling a Single-Lane Track

In order to define a path by adding B-Spline objects to the track a 3D environment is needed which provide full access to the scenegraph. Therefore the Unreal Engine is chosen.
To rebuild the test environment which is used for student projects at the FZI branch office in Berlin a 3D single-lane track, walls and some obstacles are required. To simulate a similar scene a single-lane track segment, a wall and a cube together with a cone are modeled and textured with Blender 2.79. The textures used comes from photographs taken in the laboratory of the FZI, except of the wooden texture for the cube and the cylinder that is taken from the Unreal Engine 4.18 material suite.

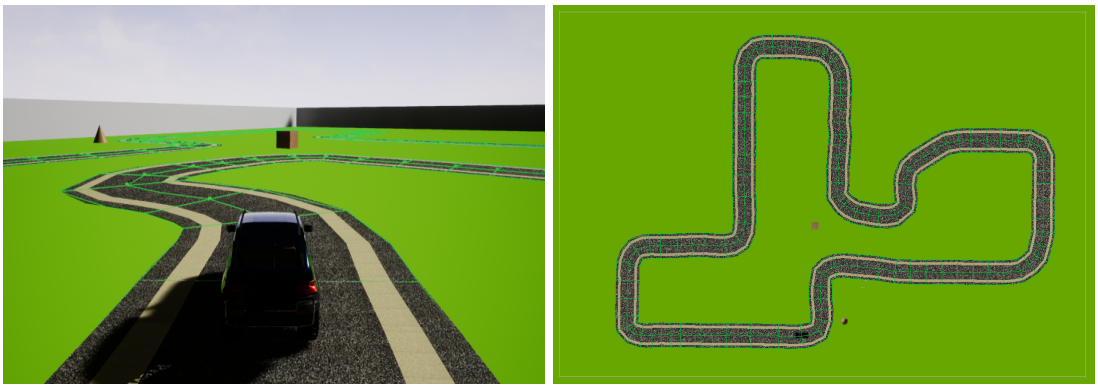


Figure 15: Scene and top-view of the single-lane track within the simulation environment.

The entire test environment build in Unreal Engine 4 can be seen in Figure 15. Herein, the green markings represent the underlying B-Splines describing the single-lane track,
which is created using Unreal's landscape spline tool. The landscape B-Spline is defined by manually selected control points, which are linked to the single-lane segment created in Blender. The landscape floor is textured with a photograph of the carpet from the FZI laboratory. To prevent the car from moving too far away from the track

walls are placed around. To use AirSim in this environment it has to be downloaded from the GitHub repository and build with the "x64 Native Tools Command Prompt for VS 2017". After that, the generated "Unreal/Plugin" directory can now simply dropped to any environment. The Plugin will be loaded by switching the GameMode setting to GameModeAirSim. Now the following section describes the structure of AirSim and how the API is extended by the calculation of the steering angle.

4.2 Extending AirSim

With the test environment described in the last section I can access the B-Spline of the single-lane track and apply the calculations from Chapter 3. As mentioned in Chapter 3.4 AirSim provides the function "moveToPosition" for multirotor vehicles, but not for the car model. To provide a similar function for the car, a path defined as a list of points to be driven, a steering angle, to adjust the orientation of the car and a velocity are needed. Regarding the previous assumptions the velocity to be driven is set to a constant value and the steering angle is calculated for each target point of the path. The following section shows the structure of AirSim and explains where the Api was extended, to realize this functionality.

Structure of AirSim

In order to implement the concept, the framework has to be extended by the following features:

- The calculation of the steering angle, using the geometry of the carpawn and the B-Spline that consists of targetpoints from the path,
- extend API by a function to calculate steering angle,
- log position, steering angle related to a camera view and the offset between the car and the targetpoint,
- extend API by the functionality to start and stop logging.

Regarding the mentioned points, the modular structure of AirSim has to be considered. At the highest abstraction layer the simulator consists of the vehicle model, the physics engine and the simulation environment to compute realistic vehicle movement, shown in Figure 16. Additionally, the simulator has a sensor module where all available sensors are implemented such as like IMU or GPS. The sensors mimic real world sensor behavior for example position errors or sensor drifts. The biggest component is given by the nearly photorealistic rendering via the Unreal Engine 4. This is required to apply computer vision algorithms for example to perform a preprocessing on scene captures that are used to train a neural network. All information from the sensors, vehicle parameter from the physics engine or scene captures can be recorded by the data collection component. AirSim also enables Hardware-in-the-loop (HIL) and Software-in-the-loop (SIL) simulation, where the hardware controller can interact with the simulator and further decision making modules for example to follow a

path. The highlighted modules in Figure 16 are all those which have been extended or adapted for the required functions. A technical description of the modules can be found in [28].

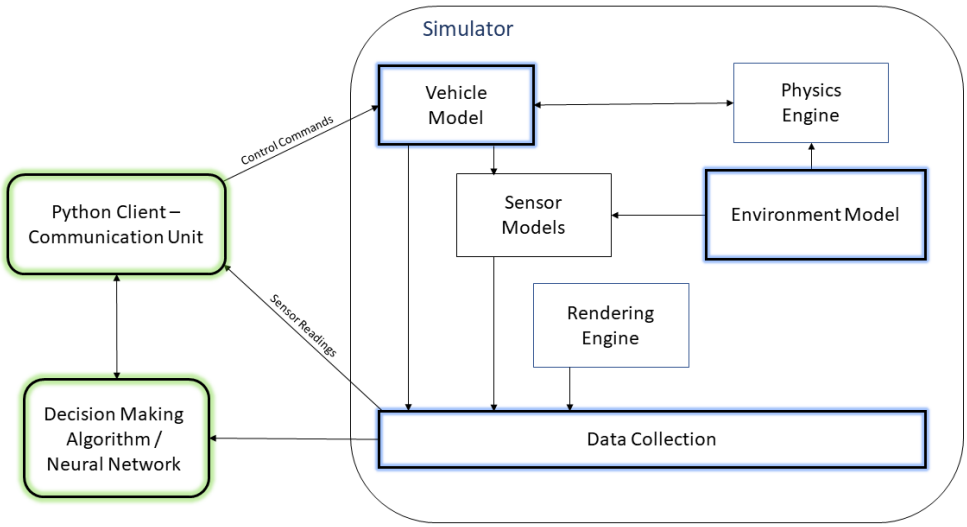


Figure 16: High level view of modular structure of AirSim and Unreal

Now, the changes made to the modules of the simulator will be explained in detail which are marked in blue.

Environment Model

The changes made to the environment model includes the modeling of the 3D scene, by importing the created 3D models from Blender. The single-lane track is build by using the Landscape Spline tool from Unreal which maps the road segment to a spline. Settings for the scene like camera defaults, time of day (position of the sun) or specific vehicle options can be set in the settings file which is loaded when starting the simulator and does not need to be integrated.

Vehicle Model

The vehicle model contains the available player characters, such as car and multirotor model. Each player has a Pawn class which is the physical representation of the vehicle in the scene. This class determines also how the vehicle interacts with the world. The car model needs an additional movement component together with a class to define the behavior for the front and backwheels. The CarPawn class is extended by the mathematical functions presented in Chapter 3.4. The CarPawn class allows to access the wheels to calculate the linear functions through the backwheels and the middlepoint between the front wheels. To access the splinepath an iteration over all

objects in the scene is required to find controlpoints which defines the spline. With access to the spline the function *FindLocationClosestToWorldLocation* can be used. It returns the closest vector (target point) on the spline path based on a given point (middlepoint between front wheels). With these points the steering angle can be calculated. Hence, the calculation of the weighted steering angle is done in python, the function to calculate the ideal steering angel is added to the API.

Data Collection

In order to evaluate the quality of the generated data it is required that images, the position and the offset are recorded to file. For that, AirSim provides a module to record data from the other modules, for example the sensor information to a CSV log file. Additionally, the settings file defines which camera and image type is used to capture scene image data such as scene view, segmentation or depth view. Furthermore, it is possible to define if data shall be recorded, even if the position and orientation has not changed and how many seconds should pass at least between the individual records.
To increase the variety of logged images three cameras will be used to capture the scene view by the left, center and right front camera. Furthermore, it is defined, that the data will only be logged if the vehicle changes position and orientation. The images of the scene are saved in the compressed PNG format.
In the Record and AirSimSettings class additional parameters can be defined, which are written for the respective vehicle into the log file. These classes are extended by writing additional values to the log file such as position, orientation and the steering angle of the car as well as the offset to the target point into the log file. The recording of the data starts when the R key is pressed. To automatically starts recording while data is generated, the functions to call start and stop recording are added to the API.

With the 3D driving environment and the calculated steering angle training and test data can now be generated. The next section describes the developed graphical user interface to easily generate data. Further, it is explained how the calculation of the weighted steering angle is implemented.

4.3 Generating Training and Test Data within the Simulator

With the possibility to calculate the steering angle to a given point, the car can follow any given path. According to the use case and the neural network to be trained, the appropriate training and test data can be generated with the announced modifications within the Simulator. For a better usage of the data generator a graphical user interface (GUI) based on WPF is implemented, shown in Figure 17.

To use the generator only the setup.exe needs to be called. All related packages such as python files or simulation environment in Unreal will be installed automatically bei the setup file. When generating the data you can choose between the following three modes:

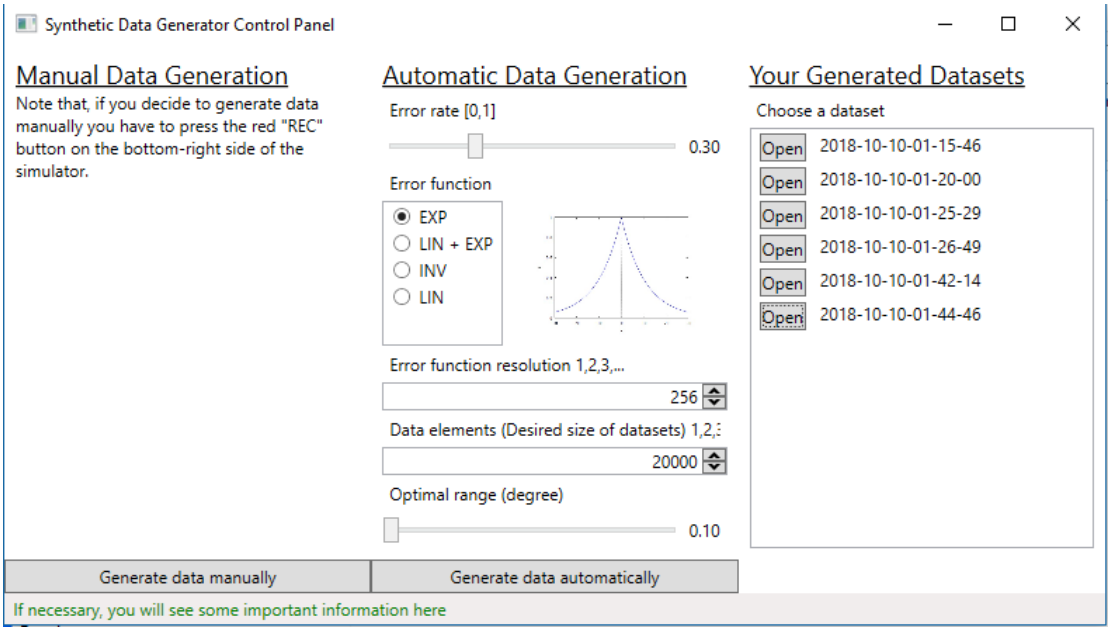


Figure 17: Graphical user interface to generate data in 3 modes.

- manual mode
- automatic mode without errors
- automatic mode with error rate in steering angle

Regardless of the selected mode a setting file will be loaded first which defines the recording intervall by 0.001 and the used cameras. The record interval defines how much seconds elaps between the frames.The differences between the individual modes are explained below.

Manual Mode

By chosing manual mode, the simulator will start and the user controls what kind of situation is recorded. This is done by pressing the record button in the simulaor or the R-key that also triggers or stops the recording of the data. The vehicle is controlled via the keyboard or the connected gamepad such as X-Box Controller.The direction of the recorded data is displayed within the simulator.

Automatic Mode without Errors

To drive without any errors the error rate have to be set to 0. Further, it is required to define the number of images that shall be recorded. By pressing the button "Generate data automatically" the python class to weight the steering, the simulation environment with AirSIm and Unreal are loaded. The recording starts automatically and continue until the data set size is reached.
While the data is generated, the calculated steering angle is sent directly to the car

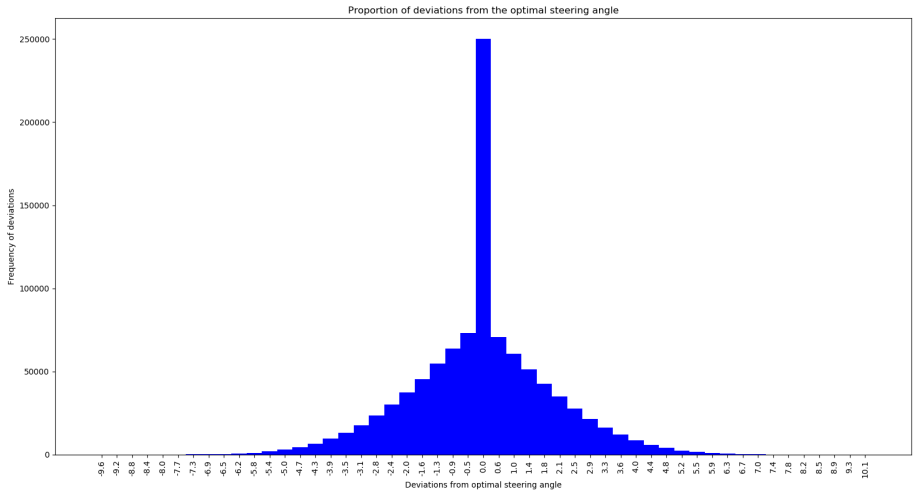


Figure 18: Proportion of the deviation from the optimal steering angle by the linear function $\pm\frac{1}{30}x + 1$

together with a constant speed value, which is reduced by 1-2 km/h in curves due to physics related steering behaviour. After finishing the data generation a graphic is shown, which displays the calculated weighted steering angles. Zero means, that the steering angle does not has changed.

Automatic Mode with Error Rate in Steering Angle

This mode only differs according to the transferred parameters. The user could choose between four different distribution functions that are explained in Chapter 3.5). Further, the input error-rate and the resolution of error-rate can be set together with the range of optimal steering angle. This means the interpretation of minimal divergences that very close to the optimal angle and therefore also interpreted as optimum.Figure 18 visualize the distribution of calculated offsets which are added to the calculated steering angle by using an error-rate of 75% and linear function.

Depending on the distribution function and the error rate, the car may deviate from the road. In this case, the user can define how far the vehicle may move away from the destination point at the middle of the road before being reset to a point. During the ride, positions that are very close to the destination point are stored in a list. If the car leaves the road or after a certain time has elapsed, a position from which the car restarts is randomly selected from this list.

This chapter has described the software architecture of airsim and how it is used to generate data for autonomous driving. The taken modifications to implement the concept presented in Chapter 3 were explained together with a detailed description how the data generator works and which parameter are needed for the different mode. Now, the data generator is able to produce a dataset with a defined size. Therefore

the following chapter adresses the evaluation of the generated data by training a convolutional neural network to drive autonomously.

5 Applicability of Automatically Generated Data to Train a Neural Network

This chapter deals with the applicability of automatically generated data in comparison to manually collected data for the training of neural networks. First, the evaluation approach is described in Section 5.1 followed by Section 5.2.1 which describes the convolutional neural network (CNN) model developed by NVidia. This model predicts a steering angle in an end-to-end manner based on a single image of the car's middle front camera. It is described how the trained model is connected to the AirSim car client to let the car drive autonomously in the scene. Finally, the last section contains the description of the experimental setup and the evaluation of the training results on the single-lane track.

5.1 Evaluation Approach

In order to evaluate the quality of the trained model, the steps shown in Figure 19 have to be followed before.

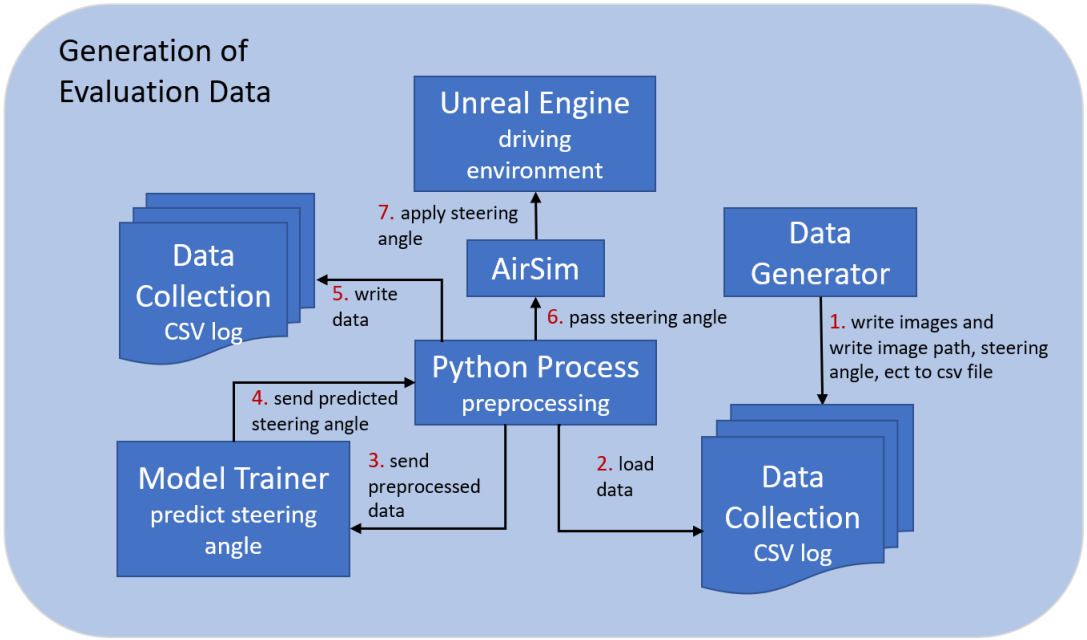


Figure 19: Process to collect data to evaluate the quality of the trained models.

The data generator announced in Chapter 3.1 is used to generate a dataset automatically and to collect data while driving manually around the track. The datasets are loaded from a Python script which preprocesses the images into the appropriate format for the trained model. These images are then forwarded to the model trainer described in Section 5.2 which contains the pretrained model to get the predicted steering angle. The results and the related input image are written to an additional csv file. These results are used to evaluate the quality and applicability of both datasets.

Step 6 and 7 contains the application of the predicted steering angle to car within the driving environment.

Subsequently, a convolutional neuronal network (CNN) developed by NVidia is trained once with the automatically generated and once with the manually retracted data set. The trained models will be evaluated by their loss rate, the number of necessary epochs until no significant improvement can be detected, measured over five epochs.

With the aim to generate evaluation data, a suitable neural network is required first and described in the following section.

5.2 Integration of CNN Model to Simulation Environment

In the process of data generation we captured the camera scene view to collect a high amount of data to train the neural network. Now we need to directly request the images from the AirSim car client to get the predicted steering angle from the model. The following introduces the CNN architecture briefly and will be followed by the taken steps to connect AirSim and the neural model.

5.2.1 CNN Architecture

In the field of image recognition, convolutional neural networks are widely used to extract features [23], [33], [18], [31]. These powerful deep learning algorithms are specialised for image processing and produces the best results.

The architecture of the neural network was developed by NVidia[4] and is used successfully in projects of Udacity, whose use case is very similar to the approach of this work.
The architecture shwon in Figure 20 consists of nine layers, including a normalization layer, five convolutional layers and 3 fully connected layers. The RGB input images is converted to YUV planes of size (66 x 200) and passed to the network. With the lambda layer from keras, the input is normalized between -1 and 1, which results in a better accuracy for the output.

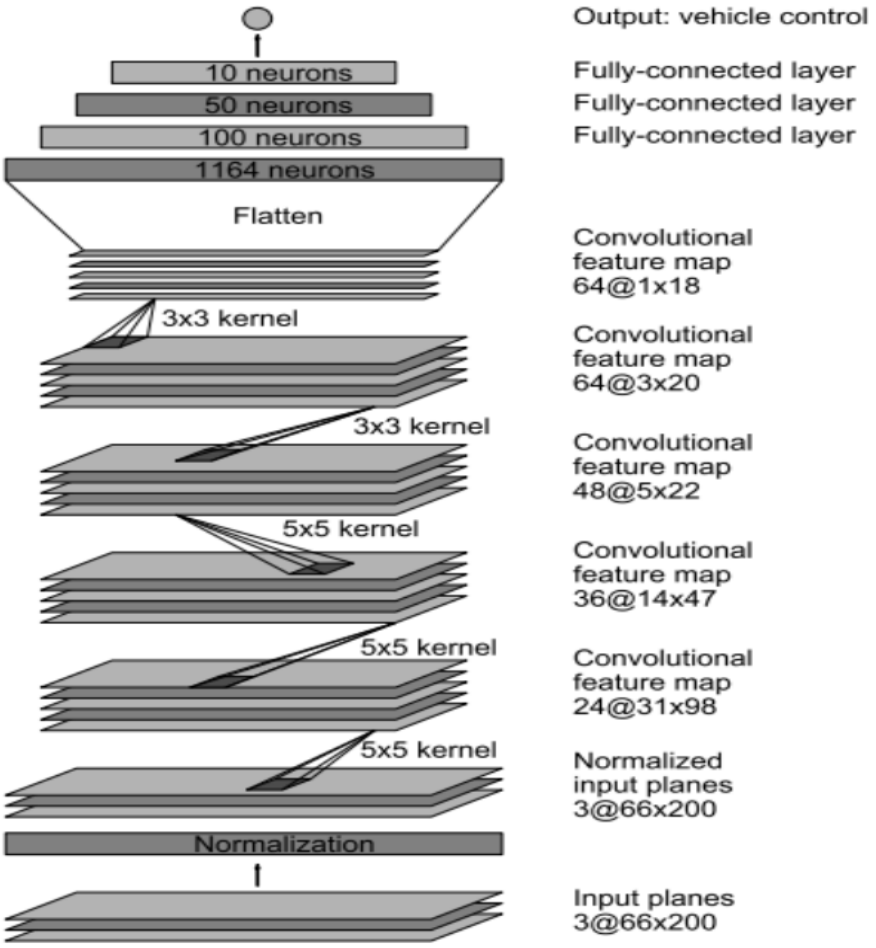


Figure 20: CNN architecture of NVidia's model

5.2.2 Training of CNN within Simulated Environment

With the generated data sets listed in the following section, it is possible to train the previously introduced CNN how to steer. Therefore the paths of the image data and the corresponding steering angles are loaded from the CSV file. In the next step, the parameters for the number of epochs to be trained, the size of the samples and the learning rate are defined. It is also passed how large the ratio of test data should be. This value is used to divide the data set into training and test data using the keras $train\_test\_split(dataset, test\_size)$ function. After that, the generator for the training and test data can be initialized. This step includes the preprocessing and augmentation, which are described in detail in the following.

Data Preprocessing and Augmentation

In order to make the neural network more robust against distortion or light effects, the input data is processed beforehand with computer vision algorithms.

At the beginning the data is mixed by a random value and then stored in a list with the pictures and a list with the corresponding steering angles. Before the augmentation takes place, some preprocessing steps are required.

- The dataset consists of three cameraviews left, center and right with the corresponding steering angle for the center view. If one of the other camera views is selected, the steering angle will be adjusted by $\pm$ 0.02.
- To prevent the neural network from extracting the color of the road environment as a feature, the color is randomly selected for each training image in the RGB space.
- Convert image to RGB

In the following it will be decided if and how the image will be augmented. You can choose between 4 options, that are shown in Figure 21:

1. No Augmentation
2. Flipp the image vertical
3. Convert image from RGB to HSV and adjust brightness
4. Shift the image by a random value and adjust the related steering angle

After the augmentation, the image is cropped to remove most of the sky and parts of the car. Furthermore, the size is changed to the default value from NVidia, which is 200 x 66. In the last step, the image is converted into the YUV space to make it even more robust against color.

With the preprocessed images, the next step is to load and configure the CNN model from Section 5.2.1. When configuring the model, the Adam Optimizer and the loss function are passed. After creating a checkpoint the training of the CNN model with the preprocessed data starts. The training continues until no significant improvement can be seen between the epochs.

5.2.3 Autonomous Driving within AirSim Environment

After the neural network has been successfully trained with the generated data, the trained model and the AirSim environment are linked. The AirSim client requests the images from the center vehicle camera, preprocesses them and passed them to the model in order to predict the corresponding steering angle. The following pseudo

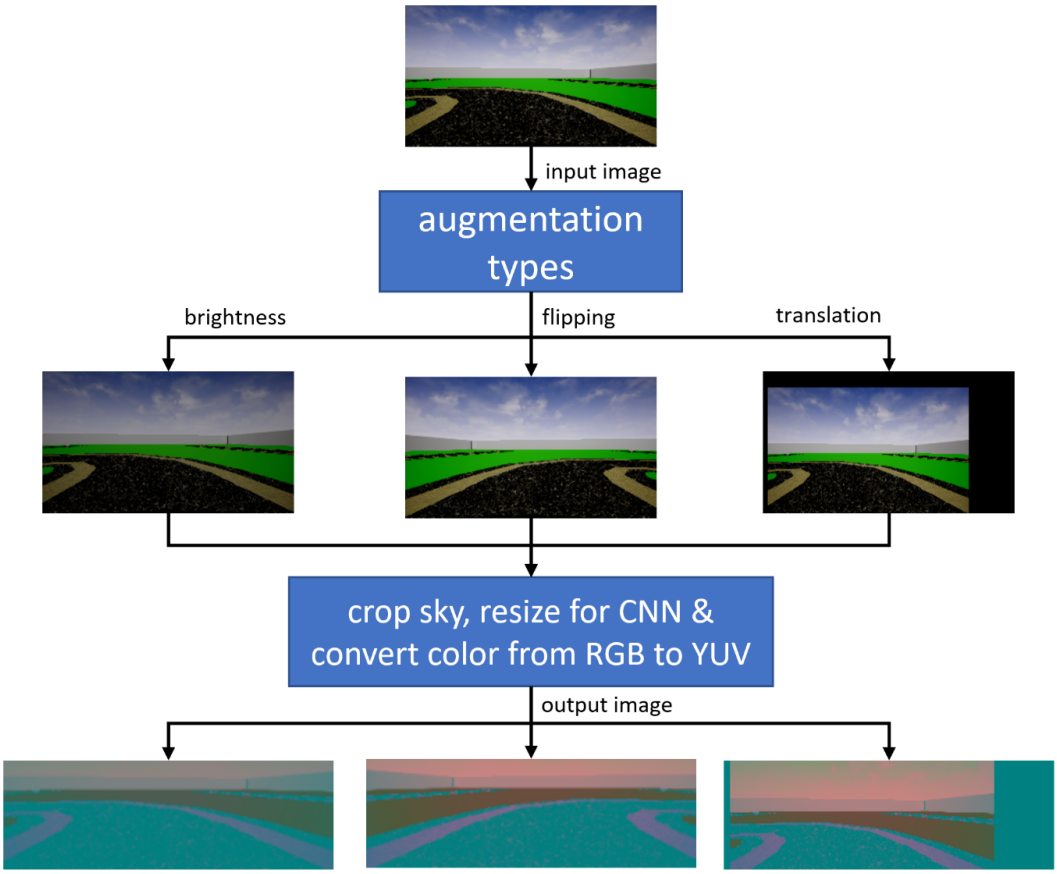


Figure 21: Preprocessing and augmentation types.

code shows the individual steps taken to predict the steering angle.

Data: images from AirSim client
Result: predicted steering angle
connect to car client;
$center_image$ = get center view image from car client;
model = load(trained keras model);
$calculated_steering$ = get calculated steering angle from scene;
while *receiving* $center_image$ *from car client* **do**
    convert $center_image$ from string to numpy array of ints;
    reshape image to 4 channel RGB image.reshape($center_image$.height, $center_image$.width, 4);
    img = cropp($center_image$);
    img = resize $center_image$ to 200 x 66;
    $preprocessed_img$ = convert RGB2YUV(img);
    $predicted_steering$ = model.predict($preprocessed_img$);
    send predicted angle, throttle to car client;
    save predicted angle and image path to CSV file;
end

Algorithm 1: Algorithm to predict steering for AirSim car client

At the beginning a connection to the AirSim Car client is established to request the data from the center vehicle camera in compressed PNG format. In the next step the trained keras model is loaded and the received images are pre-processed. The received data from the AirSimClient is available as a string, which is why a conversion into an int array has to take place first. Then the data can be further processed using the Numpy library. Since the neural network was trained on 4-channel image data in the YUV space with a certain size, the input data is brought into the same format. Then the image is forwarded to the network to predict a steering angle and send it together with the speed value to the car client. In the last step the predicted angle, the calculated angle from the scene and the image path are written into a CSV file.

5.3 Experimental Setup

To evaluate the suitability of the generated data, four datasets are created in automatic mode and one manually while driving within the test environment. The goal of the training is to learn how to drive near the center line. To achieve this, the exponential distribution function $1.1^{\pm x}$ and the linear function $\pm\frac{1}{30}x + 1$ are selected in automatic mode, which prefers small deviations from the steering angle and makes large errors unlikely. In addition, the error rate is set to 30 % and the position of the car is reset when the distance from the middlepoint of the front axle to the centerline, referred to in the following as offset, exceeds 30 cm. This value was determined from the scene and corresponds to driving over the road marking with one of the front wheels. The position at which the car is resetted is the last position at which the offset had a value below 3 cm, which corresponds to a minimum deviation from the centerline. Figure 22 shows an example distribution of the applied steering angles to the car.

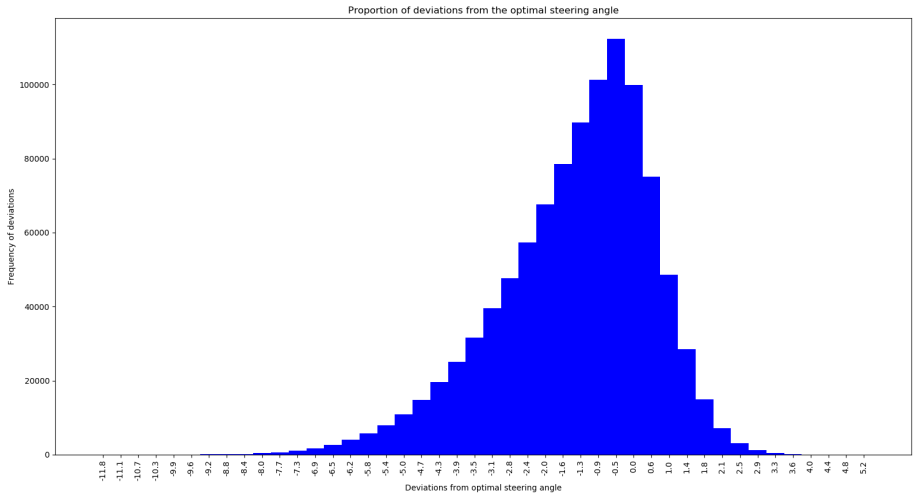


Figure 22: Distribution of applied steering angles within automatic mode

The self-entered data set was also created with the goal of moving as close as possible to the centerline. An attempt was made to keep the speed of the car as close as possible to the constant value of the automatically generated data set. Both data sets were generated until the number of recorded data points reached 20000. This corresponds to about 6 laps on the training track with a recording interval of 0.001 s between recordings. Since the course does not vary within the laps and does not have a complex environment, a larger data set would not lead to a better result.

5.4 Interpretation of the Results

This work should show that data for training neural networks can be generated automatically in a simulated environment. This has the advantage that the time-consuming manual retraction of the data is no longer necessary. Data-hungry deep learning methods can train in a sufficiently complex 3D environment until the application can be deployed on real vehicles without any risk.

In chapter 4 the developed GUI for generating the data was presented. It was used to generat four datasets automatically and a comparison dataset was entered manually. The automatically generated datasets contains a linear and an exponential distribution function with a varying error rate of 30 % and 90 %.
The data was collected on a Windows 10 Professional operating system with 16 GB RAM, an intel i5 processor and a Geforce GTX 970 graphics card. Subsequently, a model was trained for each dataset. Table 4 summarizes the core data of the training.

The records in Table 4 all show the same behavior except for the last one. The model is trained, but the value from epoch 23 hardly decreases and the training is stopped early. This can be attributed to the size of the data set. The last data set comprises

Table 4: Summary of the trained models and the related datasets.

generation method	number of images	loss improvement	validation loss improvment	trained epoch	error-rate in %
manually driven	20220	0.0972	0.0279	11	-
automatic	20000	1.24	0.926	45	30
automatic	20000	0.9175	1,8317	24	90
automatic	20000	1.3067	1.1701	23	90
automatic	66628	0.08171	0.05075	23	30

66628 images and is the only data set that runs through all 110 epochs with a steady improvement. The model drives the whole course correctly without any errors and shows no negative behavior, which occurs in the training data. Finally, it can be said that with the presented data generator large amounts of data can be produced effortlessly, which are also suitable for the training of autonomous vehicles as shown by the example of the larger data set. It has to be considered, however, that especially with the CNN selected, high contrasts between the road and the environment must be present for a successful training. In addition, the color of the road environment was randomly changed in each image, because otherwise the neural network would have paid attention only to the color and not to the road marking.

6 Summary and Future

This chapter deals with the challenges of this work and summarizes whether they have been met. Finally, an outlook is given on how this work can be used and further developed in the future.

6.1 Summary

The aim of this work was to develop a data generator that automatically generates data within a simulation environment. An evaluation of common simulation frameworks should be done to derive criteria. Based on these criteria a suitable framework should be selected. The effort by collecting driving behavior data while a human controls the car through varying scenarios, should be reduced. An approach was to be developed which defines how the vehicle can move correctly in the scene without prior instruction in order to generate appropriate data consisting of ideal and undesirable driving behaviour. After that the generated data should be evaluate by training a deep neural network.

In chapter two the dependency of training data and machine learning algorithms was pointed out in order to derive requirements for the simulation framework. Furthermore, related work was presented that dealt with questions concerning autonomous driving in simulators. Since the requirement was to choose a simulation environment that supports photorealistic rendering to reflect scenarios from the real world, AirSim was chosen. In the following chapter, an approach was presented that allows to follow a path with the car in AirSim. This is based on the generation of a spline object which assumes the middle of the road as the destination. Due to the requirement to use alternative routes on the road, a method was developed which is based on the calculation of a weighted random value which is added to the calculated steering angle. It was shown that by using different distribution functions, alternative paths can be driven bei the car in frequency. In order to allow a simple usage a GUI was introduced in Chapter 4 which can generate data automatically by providing some parameter. The last chapter dealt with the evaluation with the CNN of NVidia. It could be shown that with little effort and low error rates the complete track could be driven autonomously by the car. But with increasing error rates, the training effort also increases.

All in all it was shown that it is possible to train autonomous driving without manually collected data. Furthermore, a more robust network can be trained on the basis of synthetic data by using well chosen distribution functions.

6.2 Future Work

This work provides the basis for many possible applications, because of the chosen framework AirSim. It is constantly evolving new features and funtionalities such as the control of several vehicles in the scene. Regarding the developed approach by

varying over the steering angle instead of calculating trajectories a lot of scenarios can be researched.For example, if a suitable function is found, a certain driving behaviour can be simulated with it. By using slight deviations, the vehicle has become more robust against this type of noise. On the other hand, frequent large deviations have caused the training to take longer. Therefore, a work that specifically evaluates mathematical function to map driving profiles would be a great benefit.

Furthermore, the data generator can be used in smaller student projects from the FZI. For example, a realistic simulation environment can be set up to provide data for the training of the neural network with a direct connection to the model vehicle. Therefore the implementation can be extended by an obstacle avoidance based on machine learning algorithm.

As the development presented here is based only on a circuit track that is driven on the basis of the steering angle, an extension via additional control commands such as braking, accelerating or reversing would also be conceivable.

Bibliography

[1] AirSim. References. https://github.com/Microsoft/AirSim/blob/master/docs/who_is_using.md, sept 2018.

[2] autonomos labs. Madeingermany. http://autonomos-labs.de/, August 2018.

[3] Berniw. Torcs racing board platform.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *arXiv:1604.07316v1 [cs.CV]*, 04 2016.

[5] A. Seff C. Chen, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of 15th IEEE International Conference on Computer Vision (ICCV2015)*, 2015.

[6] Jui-Jen Chou and J.J. Tasy. Trajectory planning of autonomous vehicles in environments with moving obstacles. *IFAC Proceedings Volumes*, 26(1):439 – 445, 1993. 1st IFAC International Workshop on Intelligent Autonomous Vehicles, Hampshire, UK, 18-21 April.

[7] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *International Conference on Robotics and Automation (ICRA)*, 2018.

[8] coppeliarobotics. V-rep - virtual robot experimentation platform. http://www.coppeliarobotics.com/, October 2018.

[9] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[10] Xiaohui Dai, Chi-Kwong Li, and A. B. Rad. An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems*, 6(3):285–293, Sept 2005.

[11] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[12] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA, an open urban driving simulator. http://carla.org/, 2017.

[13] Unreal Engine. Unreal engine, May 2018.

[14] Epic Games. Unreal engine 4. https://www.unrealengine.com/en-US/what-is-unreal-engine-4.

[15] gazebosim. Gazebosim robot simulation. http://gazebosim.org/, October 2018.

[16] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, June 2012.

[17] Vladimir Haltakov, Christian Unger, and Slobodan Ilic. Framework for generation of synthetic ground truth data for driver assistance applications. In Joachim Weickert, Matthias Hein, and Bernt Schiele, editors, *Pattern Recognition*, pages 323–332, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[18] Kaiming He and et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. arXiv preprint arXiv:1502.01852, 2015.

[19] Tom Hirschberg and Dean Zadok. Self driving simulation for autonomous formula technion project. http://www.cs.technion.ac.il.

[20] Xinyu Huang, Xinjing Cheng, Qichuan Geng, Binbin Cao, Dingfu Zhou, Peng Wang, Yuanqing Lin, and Ruigang Yang. The apolloscape dataset for autonomous driving. *CoRR*, abs/1803.06184, 2018.

[21] Nidhi Kalra and Susan M. Paddock. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?, 2016.

[22] J. Kim and C. Park. End-to-end ego lane estimation based on sequential transfer learning for self-driving cars. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1194–1202, July 2017.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks., 2012.

[24] NVidia. Homepage. https://www.nvidia.com/de-de/, oct 2018.

[25] Playment. Data labeling for computer vision. https://playment.io/, oct 2018.

[26] Jörnsen Reimpell, Helmut Stoll, and Jürgen W. Betzler. Preface. In Jörnsen Reimpell, Helmut Stoll, and Jürgen W. Betzler, editors, *The Automotive Chassis (Second Edition)*, pages 266–306. Butterworth-Heinemann, Oxford, second edition edition, 2001.

[27] Waymo Safety Report. On the road to fully self-driving, August 2018.

[28] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Aerial Informatics and Robotics platform. https://www.microsoft.com/en-us/research/wp-content/uploads/2017/02/aerial-informatics-robotics.pdf.

[29] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[30] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim programmatic control. https://github.com/Microsoft/AirSim, 2018.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[32] sourceforge. Maxima a computer algebra system. http://maxima.sourceforge.net/, October 2018.

[33] Christian Szegedy and et al. Going deeper with convolutions, 2014.

[34] Udacity. Udacity dataset mountain view. https://github.com/udacity/self-driving-car/tree/master/datasets, August 2018.

[35] Joe Venzon. Vdrift. http://vdrift.net/, March 2018.

[36] waymo. waymo. https://waymo.com/, August 2018.

[37] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, and Andrew Sumner Rémi Coulom. TORCS, The Open Racing Car Simulator, 2014.

[38] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. BDD100K: A diverse driving video database with scalable annotation tooling. *CoRR*, abs/1805.04687, 2018.

[39] Jing-Shan Zhao, Zhi-Jing Feng Xiang Liu, and Jian Dai. Design of an ackermann type steering mechanism. In *Journal of Mechanical Engineering Science*, volume 227. 11 2013.