TU WIEN Informatics

Simulated Data Generation for Machine Learning Based Driving Assistance

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Michael Rubik
Matrikelnummer 11809937

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Mag. Stefan Ohrhallinger, PhD

Wien, 21. Jänner 2023

Michael Rubik Michael Wimmer

TU WIEN Informatics

Simulated Data Generation for Machine Learning Based Driving Assistance

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Michael Rubik
Registration Number 11809937

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Mag. Stefan Ohrhallinger, PhD

Vienna, 21st January, 2023

Michael Rubik   Michael Wimmer

Erklärung zur Verfassung der Arbeit

Michael Rubik

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21. Jänner 2023

Michael Rubik

Kurzfassung

Punktwolken bilden als Daten den Grundstein für viele Machine Learning Algorithmen, vor allem im Bereich von Objekterfassung und automatisierten Fahrzeugassistenten. Solche Datensätze entstehen oft durch LIDAR Sensoren, welche an ein Auto angebracht wurden, und in der echten Welt fahren. Diese Arbeit befasst sich mit dem Generieren solcher Daten in einer simulierten Welt, durch virtuelle Sensoren. Unser Ziel ist es, den signifikanten Ressourcen- und Zeitaufwand zu vermeiden, welcher vor allem bei größeren Datensätzen und der Anpassung an spezielle Anforderungen notwendig ist. Wir verwenden den CARLA Simulator um virtuelle Daten in einer vollständig künstlichen Welt zu sammeln. CARLA ist eine Open Source Simulator für urbane Umgebungen und Verkehrssituationen, mit frei verwendbaren digitalen Modellen. We beschreiben wie gefährdendes Verhalten erzeugt werden kann und der generierte Datensatz unterschiedliches realistisches Verhalten abdeckt.

Abstract

Point clouds are data at the foundation of many machine learning algorithms, especially in the field of object detection and autonomous driving systems. LIDAR sensors produce such datasets attached to a car driving around the real world. This thesis explores the generation of such data in a simulated world through virtual sensors. We aim to avoid the significant resource and time investment otherwise needed, especially concerning larger data quantities and datasets tailored to specific needs. We utilize the CARLA simulator to collect virtual data from an entirely artificial world. CARLA is an open-source simulator for urban environments and traffic with ready-to-use digital assets. We describe how we influence the standard behavior of the simulator to enforce desired situations. We explain how behavior that may be harmful can be enforced, as well as show how the generated dataset covers the different kinds of behavior a traffic participant may encounter.

Contents

CHAPTER 1

Introduction

Autonomous Driving (AD) and Machine Learning (ML) are trending terms in the public eye and research. In recent years these fields experienced a lot of growth and attention. AD primarily addresses cars and motorized vehicles, as the aim is for such systems to fully register the surroundings and make decisions based on this information. However, these systems can also be adapted for or developed for bicycles. As cycling receives less attention, even though ML-based technologies can support it, we aim to contribute to such research.

Data, or rather training data, builds the foundation for Machine Learning (ML). In the context of traffic situations, this means attaching sensors to vehicles or pedestrians and recording a variety of scenarios under different conditions. For ML, the collected data needs to be processed, the sensors synchronized, bounding boxes drawn, and objects labeled. With the ever-growing field of ML, primarily due to AD, challenges regarding this early step in the process arise. Complex models need datasets of such a large scale that cost and time factors regarding data acquisition require more attention. Although there are established datasets created for such purposes, they often need more size and quality or are limited in the type of data collected. Therefore the need for new datasets remains. In these previous projects, the sensors used were expensive, and the recording could only be executed during certain conditions. The collected data then needed to be manually annotated for accurate ground truth. This process is relatively trivial albeit resource-heavy, especially if compared to the more complex ML models that rely on them.

However, not all data has to come from the real world, and computers offer increasingly performant virtual worlds and simulation tools. As a computer generates the whole synthetic world, every piece of information inside it is also available and can easily be extracted. The problem of labor-intensive ground truth is addressed, and simulation runs can also be repeated and adapted as needed to generate desired quantities and situations.

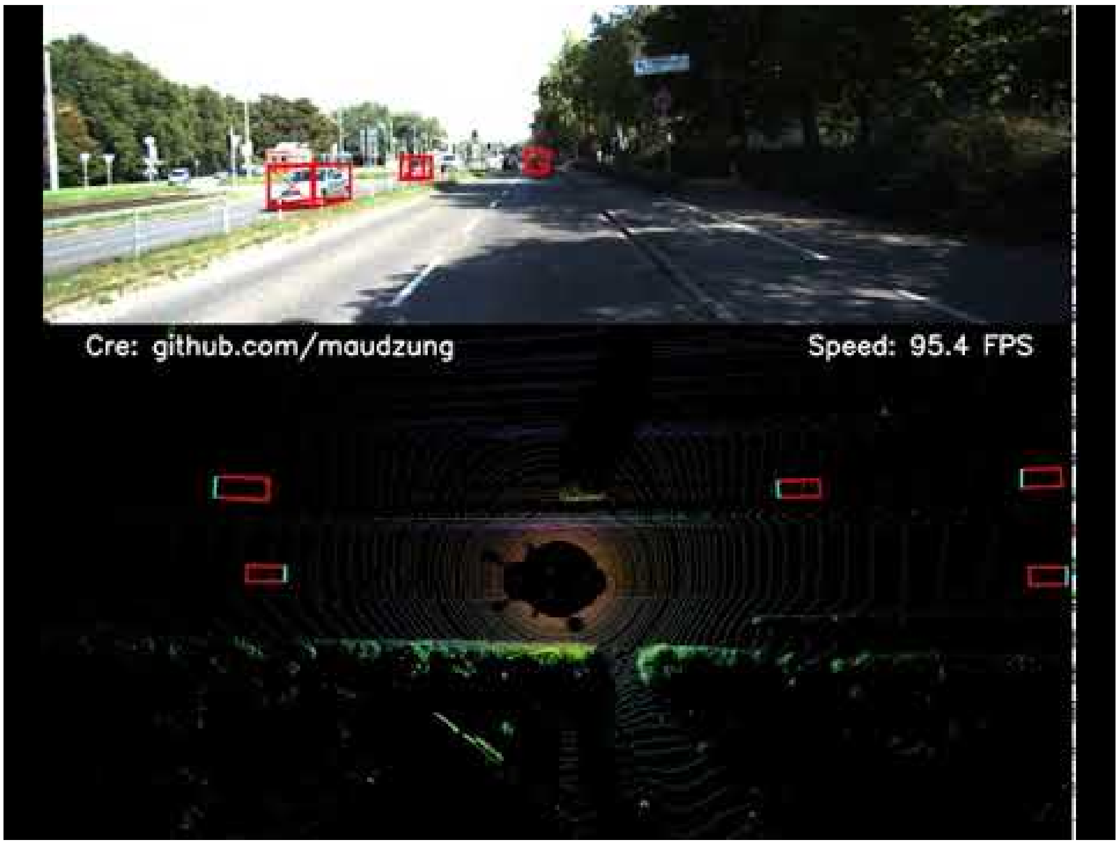


Figure 1.1: Object detection based on point clouds [Dun20].

The CARLA Simulator is one such environment for AD research. It is an open-source tool that combines source code and digital assets for developing, training, and validating AD tasks and systems. Necessary for this thesis is the support for simulated sensors, specifically a LIDAR scanner. Such a tool is costly in the real world, especially if 360° coverage is needed. ML models can interpret LIDAR data as 3D point clouds, which are then fed into an algorithm for object detection, as seen in Figure 1.1. Inside this virtual world, the simulator can spawn different vehicles, automatically controlled by a provided traffic manager, and interact with each other and pedestrians. Such situations are recorded and compiled into a dataset similar to ones generated in the real world.

This thesis aims to extend this foundation of data for ML, with a focus on data collection around cyclists. Utilizing CARLA, simulated worlds surrounding a bicycle and its virtual cyclist are created. We let the standard traffic manager take over, with minor adaptions, to drive around these worlds and record them. At the time of writing, an amendment to Austrian traffic regulations came into effect, specifying a 1,5- to a 2-meter distance for vehicles overtaking bicycles. Taking this into account, we extend car behavior regarding closer proximity or collisions to record a variety of everyday or dangerous traffic scenarios. The generated dataset can be used to train an ML model without the need to build a sensor setup in the real world or expose a cyclist to harmful traffic situations.

Our work that is covered by this thesis is publicly available under the GNU Lesser General Public License through our GitLab repository[OPR23], where the data generation script[ROP23] is located.

The following chapter will first discuss related work in the field of real-world dataset generation for ML, specifically for AD algorithms, to present a view on the state-of-the-art. Then it will talk about synthetic datasets, as this field is newer and uses real-world datasets to compare. Chapter 4 describes our approach to generating the data and how desired behavior is enforced. To conclude, we evaluate the data regarding the variety of situations covered and discuss how future work may expand our method and how further research may build on it.

CHAPTER 2

Related Work

In this chapter, different and similar approaches to the generation of data from traffic scenarios will be discussed. Especially in the context of machine learning (ML) and autonomous driving (AD), big and unbiased datasets are needed. Besides the quantity of data, also certain situations are relevant. Such cases include closer proximity to the recording vehicle, measurements regarding the spread of directions of traffic, or generally some form of finer control of certain aspects.

There are already some datasets available, usually to be used as a foundation for AD. Their generation and post-processing are resource intensive, as the sensors are expensive, and measured data needs to be manually annotated.

Data generated in synthetic environments are proposed [Nik21] as a substitution for or complement to real-world measurements. For this thesis, especially data from LIDAR sensors are relevant. These provide the ability to model the environment as a high-density point cloud, well suited for ML in traffic situations.

2.1 Real World Datasets

The evident approach is to mount the required sensors onto a vehicle and drive through various streets and regions to collect the data. The advantages are realistic modeling of the measured situations in the real world and capturing human behavior in traffic.

2.1.1 The KITTI Autonomous Driving Project

Geiger et al. [GLU12, GLSU13] provided a novel dataset for visual tasks in the context of autonomous systems. The aim was to generate a benchmark for visual odometry / SLAM, stereo / optical (scene) flow, and 3D object recognition. GPS positions, annotations, and 3D bounding boxes serve as ground truth. A station wagon was equipped with a sensor

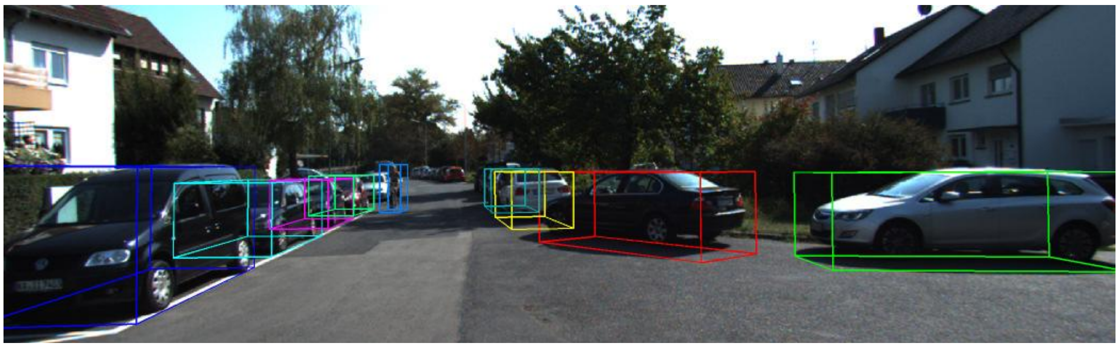


Figure 2.1: Each frame of the KITTI dataset [GLU12] was annotated with bounding boxes for each vehicle, pedestrian, and cyclist. These serve as ground truth for object detection algorithms, usually with point clouds as input.

array containing 180° coverage through cameras in color and grayscale, a Velodyne HDL-64E 3D LIDAR scanner for 360° environmental view, and a device for GPS and inertia measurements. These sensors were cross-calibrated and synchronized semi-automatically to obtain reliable ground truth. The car was driving around Karlsruhe, Germany, where different environments like freeways, rural and metropolitan areas were observed, and various weather conditions. State-of-the-art algorithms were run on their various subsets of data generated to evaluate their performance and compare them with previous, smaller datasets of either less quality or quantity. For this thesis, particularly the scene flow and object detection results were relevant as these utilize point clouds.

In the experimental evaluation, object detection had a high precision on weakly and not at all occluded objects. However, it is mentioned that these only apply to less than 20% of the 4 000 objects in 12 000 images. The results of models in stereo matching were compared to their rankings on an established dataset [SS02], with the observation that previously high-performing algorithms now had bad results. In optical flow, even the best method applied to the dataset had an error rate of 11%. Geiger et al. conclude that previous datasets provided insufficient data for training, resulting in overfitting, and suggest that in some cases, more complex models that rely on larger training sets are needed. The KITTI project became a frame of reference for many future datasets [COR+16, WWY+21, HWC+20, BGM+19, NOBK17] and a benchmark for AD-related methods and algorithms [FKG13, MG15]. As it is the first of such scale, and because of its various references, it remains an important milestone often used in the evaluation or test phase of new projects.

KITTI-ROAD

Fritsch et al. [FKG13] utilized KITTI [GLSU13] as the foundation of the KITTI-ROAD dataset. They extracted 600 frames with at least 20 m spatial distance and low traffic density. They were divided into different road categories and into training and test data. KITTI-ROAD aims to offer a benchmark for road detection algorithms, with proposed evaluation methods for the road area and lane of the vehicle.

Menze and Geiger [MG15] proposed a novel method for scene flow, where the layout of a scene is segmented into individual moving objects. 400 scenes from KITTI [GLSU13] were collected and annotated with additional scene flow ground truth to evaluate this model. It was found that only some of the existing models were fit for the extreme motions in some of these realistic scenes. Menze and Geiger [MG15] provide an additional example of how the KITTI AD project is a foundation for novel approaches to AD algorithms.

2.1.2 Further Datasets

As mentioned, the KITTI project set the standards for later datasets. These then expand on the size, variety, and properties of data collected and usually focus on a single task.

For example, Cityscapes [COR+16] mentioned the lack of complexity regarding the lowest level of semantic labeling in Geiger et al. [GLU12]. As mentioned in Subsection 2.1.1, the ground truth for object recognition in the KITTI dataset was established through bounding boxes. Cordts et al. [COR+16] provide fine and coarse annotations at the pixel level. The difference between these annotations is seen in Figure 2.2.

It took 1.5 hours of human labor per frame for the 5 000 frames with fine annotations and 7 minutes per one of the 20 000 frames with coarse annotations. Cordts et al. split these subsets into training, additional training, validation, and test sets. This way, a balance of geographic location, population density of individual sites, and time of year of the recording is guaranteed. A state-of-the-art semantic labeling model was applied to the Cityscapes data and achieved similar or even better results than the best-reported one if evaluated on KITTI data. Cordts et al. emphasize the mentioned advantages, as well as underscore the importance of additional datasets for cross-evaluation.

The ApolloScape dataset for AD [HWC+20] expands on both KITTI and Cityscapes. As mentioned, KITTI only provides a limited number of frames for training, and Cityscapes only has discrete semantic labeled frames without 3D information. Huang et al. [HWC+20] capture real-world scenes with stereo images, GNSS/IMU data, and

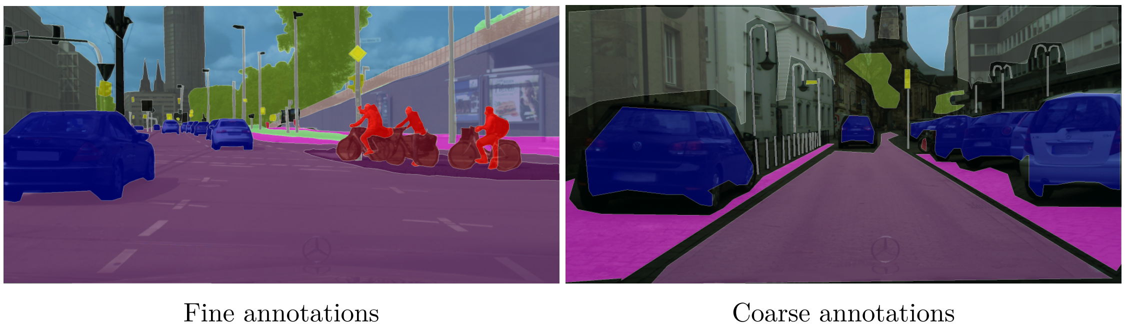


Figure 2.2: The difference between the fine and coarse annotations of the Cityscapes dataset [COR+16].

a LIDAR scanner for point cloud data. Through an efficient labeling pipeline, fine annotations comparable to Cityscapes were achieved in 30% of the time per frame. While not offering 3D bounding boxes, frames are annotated with 3D semantic points. Huang et al. collected data from different regions in different weather conditions and of higher quantity than both mentioned datasets.

2.2 Simulated Data Generation

With the obvious benefit of realism of the datasets mentioned in Section 2.1 come certain drawbacks. For example, in ML, ground truth needs to be established. In the context of semantic segmentation of 3D objects like cars and pedestrians, ground truth can be represented through bounding boxes like in Subsection 2.1.1. Cordts et al. [COR+16] address this lack of detail in the Cityscapes dataset, thereby improving it through pixel-level polygons. This did require much human labor, even more than one hour per frame, including quality control. The finer the quality of annotation or the more complex such a task is, the number of resources necessary becomes greater and greater. Another drawback is the amount of work that goes into data generation. In Section 2.1, cars had to be equipped with numerous more or less expensive sensors, which then had to be synchronized and calibrated for each trip. Apart from the margin for error, further complications arise. Each trip takes time, weather conditions have to be met, and traffic conditions may also vary. Furthermore, if a project's goal is observing certain scenarios, like crashes or near collisions, repeated trails are nigh impossible. In general, modern ML is limited by insufficient data. To avoid issues like overfitting, synthetic data is proposed as a solution [Nik21]. This way, the necessary scenario can be simulated and recorded through a computer. This enables recording situations that are hard to reproduce in real life, like crashes or the same scenario with different parameters. Labeling can be a resource-intensive part of the dataset generation as well[COR+16], and computer-generated worlds offer this information usually as a by-product of the simulation process. Data can be generated and annotated as needed, in variable sizes, and under specific conditions at any time, as just another step in the ML pipeline. The data may be entirely synthetic, often rendered in a game engine like Unity or Unreal, or semi-real, where certain augmentations were performed on real-world data. This chapter will discuss these approaches, their differences, and their findings.

Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?

Addressing the issue of dataset bias, Johnson-Roberson et al. [JRBM+16] aimed to compare the real world, established datasets of fewer images, notably Cityscapes [COR+16] and KITTI [GLSU13] to larger, computer-generated ones. The simulation was conducted in a modified version of the Game GTA V, as it provides a simulated world of high fidelity. The game engine provides bounding boxes for each object in a certain radius around the player, so this information was extracted and used as part of the ground

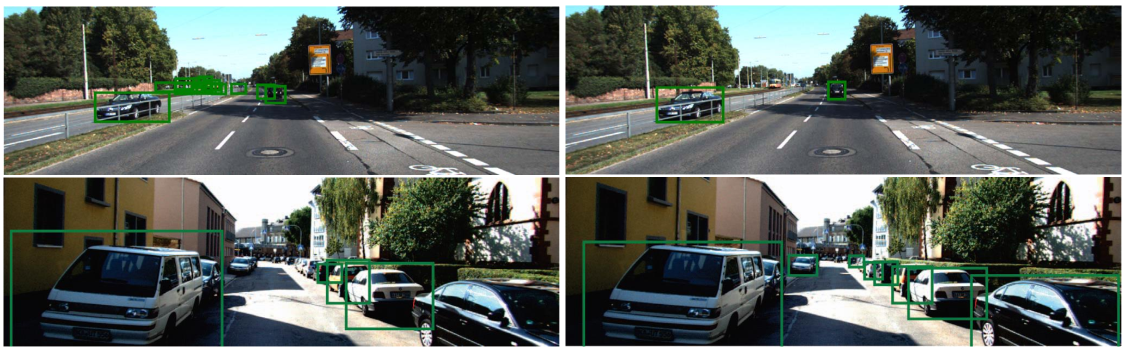


Figure 2.3: A visualization of representative detections from training on Cityscapes [COR+16] and the dataset of 20 000 simulated by Johnson-Roberson et al. [JRBM+16]. The simulated dataset produces less cluttered outputs and higher quality in bounding boxes.

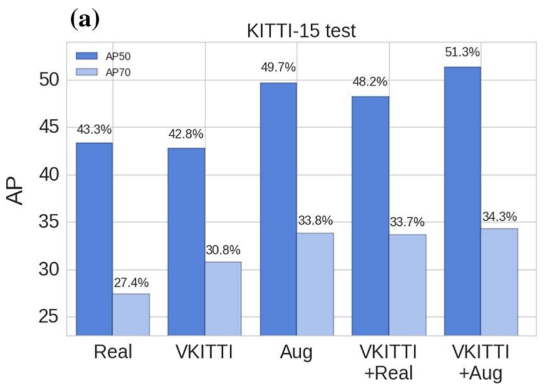
truth. Using the GPU's stencil and depth buffer from the engine, distances and labels for the objects could be computed. These annotations complete the ground-truth generation. For evaluation, Johnson-Roberson et al. recorded sequences of different lengths of frames so that deep learning object detection algorithms could train on them and the Cityscapes dataset. The same models trained on the artificial data outperformed the one real-world data-relying model. It is also notable how the outputs of the bounding boxes from the simulated datasets are of superior quality to the real-world trained network, as seen in Figure 2.3. Johnson-Roberson et al. show how data from a source that was not even intended for such a purpose can compete with real-world data due to its quantity.

Augmented Reality Meets Computer Vision: Efficient Data Generation for Urban Driving Scenes

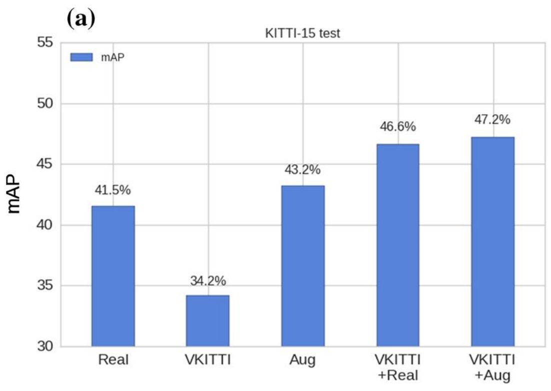
As an alternative to fully synthetic data, a combination of both virtual imagery and real-world data is proposed by Alhaija et al. [AAMM+18]. The paper aims to explore semi-synthetic data, building on the achievements of fully computer-generated datasets, and avoiding the problems of real, labor-intensive datasets. The focus was on visual detection, as there were already established datasets. Alhaija et al. selected Gaidon et al. [GWCV16] (VKITTI) as the virtual candidate and data by Liao et al. [LXG21] (KITTI-360) as a real-world candidate. These were compared to their approach.

The augmented dataset relied on 200 real-world images from KITTI-360, which were augmented by rendering other cars on top of them. A combination of manually defined rules and random placements determined the location and orientation of the additional vehicles. One such image was reused up to 20 times with five more cars in each version in their dataset, showing the best results.

As seen in Figure 2.4, the method trained on the augmented KITTI-360 dataset performs



Instance Segmentation



Object recognition

Figure 2.4: Results from the algorithms used by Alhaija et al. [AAMM+18] on the combinations of real, synthetic, and augmented data. The one trained on VKITTI [GWCV16] outperformed if the training was fine-tuned with further augmented Images as training data.

better than the virtual model. It achieves similar results to the one trained on virtual data fine-tuned with real-world images. This shows how the realism from the added cars and the real background, combined with increased dataset size, provide the best results on this data. While there are still limitations, especially regarding the placement of the cars, Alhaija et al. [AAMM+18] show a possible new approach to synthetic data.

2.2.1 The CARLA simulator for data generation

Dosovitsky et al. [DRC+17] introduce the open-source system CARLA for training, developing, and validating in fields like AD and ML. Initially, CARLA was used to evaluate approaches to AD, such as imitation learning and reinforcement learning. The simulator is designed to provide an environment where such agents can be trained and evaluated. As Dosovitsky et al. did this in a virtual world, one agent could train for up to 12 days without interruption. The CARLA simulator was also used to create systems to support development in other traffic-related ML tasks. The following section will present such tools and describe their different key aspects.

Multimodal 3D Object Detection from Simulated Pretraining

Brekke et al. [BVL19] present CADET, a dataset extraction tool working with CARLA, capable of producing datasets similar to KITTI. Focusing on 3D object recognition, a simulated camera and LIDAR scanner were attached to the recording vehicle and synchronized for accurate data. The use of the generated datasets was then evaluated against the KITTI dataset. Different object detection models were trained and evaluated on either set or a combination of both. The results show that a fully synthetic dataset may not be sufficient for training but can supplement real-world data. CADET is a tool capable of generating a dataset capable of lowering the real-world data necessary to train complex ML systems. The paper provides guidelines for a general simulation pipeline, where the variety inside the CARLA simulator is utilized to the fullest.

Train Here, Drive There: Simulating Real World Use Cases with Fully-Autonomous Driving Architecture in CARLA Simulator

The CARLA simulator offers, besides a realistic and fully autonomous driving simulation, the CARLA Autonomous Driving Challenge. This is a repository for scenarios that CARLA can easily execute. Gómez-Huélamo et al. [GHDEB+20] selects scenarios where their fully-autonomous driving architecture is evaluated. The vehicle uses the simulated data from virtual GPS, LIDAR, and camera sensors for decision-making. Is confronted with situations where the car has to face other vehicles, pedestrians, and emergency brakes. The Robot Operating System shows how the decision-making of fully automated architectures can be trained in a simulation.

CarlaScenes: A synthetic dataset for odometry in autonomous driving

Kloukiniotis et al. [KPA+22] utilizes CARLA to generate a benchmark for odometry tasks in AD and evaluate state-of-the-art methods. Thereby drawbacks of these are shown. Weak points in datasets recorded in the real world regarding visual odometry are the GNSS irregularities, especially in environments such as tunnels and urban canyons, as well as a lack of variety in different weather conditions. The CARLA simulator weather can be set through parameters at the start of each simulation, GNSS sensors provide accurate measurements, and specific scenarios can easily be repeated under different conditions. One such run was recorded through a sensor array, similar to the one used in KITTI [GLSU13, MG15], with additional modalities to collect ground truth that would otherwise need to be manually annotated.

By comparing different odometry techniques, Kloukiniotis et al. [KPA+22] evaluated the benefits of simulation data for the respective methods. Deep learning-based methods must be generalizable to different dynamic environments, and training data from CARLA provides multiple scenes. Geometry-based mapping and odometry lack in detecting abrupt changes in their environments and therefore need to be tested on dynamic environments. CarlaScenes provides a dataset to test visual odometry methods and the foundation for further improvements regarding scene variety.

CHAPTER 3

Background

In this chapter, we present the tools used in this work and how they are adjusted and used for the goals of this thesis. As the aim is to provide a foundation for object detection, the three-dimensional space is to be represented through point clouds. The first chapter will discuss how LIDAR scanners generate such data, and then the simulation environment CARLA is introduced and why we chose it. Lastly, we describe how CARLA is configured and applied to generate the resulting dataset.

3.1 LIDAR

The central piece of the dataset are point clouds, corresponding to each frame recorded. This chapter will describe how light detection and ranging scanners (LIDAR) generate such data. LIDAR generally describes a method to determine distances to an object's surface, with applications in airborne mapping, geology, meteorology, and AD [SL21]. Such a scanner repeatedly sends out laser pulses of a certain frequency, collects the reflected light, and measures the time t it takes. LIDAR systems utilizing the time-of-flight principle calculate the distance d toward the reflecting surface with the formula

$$d = \frac{c \cdot t}{2}, \tag{3.1}$$

where c is the speed of light in the current medium.

Currently, two forms of LIDAR systems are available [YDB+18]. Flash LIDAR scanners take more than one measurement out of the backscattered light and measure the full waveform. Given that one object does not fully occlude another, one flash of light will reflect from many different surfaces or many times from the same surface. The entire scene is illuminated at once, and a receiving unit matrix collects the light. In processing, the information on which the sensor received the signal can be used as spatial data to calculate the origin in world coordinates. Because the light is emitted at a wide angle, the intensity is relatively low, and therefore the maximum distance is limited as well.

Especially in AD, scanning LIDAR technology is used. A rotating laser pulse covers up to a 360° field-of-view for full coverage. The beam is focused, so it can be used at a wider distance to provide ranging of different surroundings. Information about specific sensors or the angle is used to generate points in 3D space. Each point of one frame is combined into one point cloud, describing the surfaces of objects in range and the sensor's field of view.

3.2 CARLA

In contrast to datasets generated through driving in the real world, this work focuses on an approach in a simulated environment. Dosovitskiy et al. [DRC$^+$17] introduced a unique tool to support AD research, the CARLA (Car Learning to Act) Open Driving Simulator[1]. It is an open-source project built as a layer on top of the Unreal Engine 4 with a free library of resources and assets, like cars and cities, to be used in developing AD models. The engine provides realistic physics and simple logic for actors like cars and pedestrians in traffic and renders them of state-of-the-art quality. Furthermore, different environmental parameters may be set to simulate different weather scenarios and lighting conditions.

Generally, it follows a client-server architecture, where the server handles the simulated physics, sensor rendering, and actors. On the client side, one such client module controls specific parts defined by the users. A script written in either C++ or Python can access the server and invoke certain behavior through the CARLA API. For example, a traffic manager, a built-in CARLA system that can control selected actors in the simulation. It is designed to generate realistic behavior of pedestrians at sidewalks and crossings, as well as control the movement of vehicles on the streets. Basic rules include avoidance of collisions, stopping at red lights, and adhering to the speed limit. If needed, these rules can be modified and, more importantly for this thesis, modified only for selected vehicles.

For reinforcement learning purposes, CARLA also enables client scripts to take full control of actors, without the interference of the traffic manager, for the model to train based on its decisions. For other ML purposes, especially the sensors are relevant. It is possible to attach such virtual sensors to an actor to gather data during simulation runs, effectively generating training data from the experienced situations. One such sensor is the LIDAR scanner. Through raycasting along the vertical field-of-view in a specified time frame, points correlating to LIDAR measurements are calculated and stored as locations in CARLA space.

Another feature that further suggests CARLA as the tool for this thesis is a library of open assets like cars, motorcycles, walkers, static objects, and, especially relevant, a bicycle and driver. Additionally, CARLA provides support for more specific types of

[1]For this thesis, we used CARLA Version 0.9.11: https://carla.readthedocs.io/en/0.9.11/

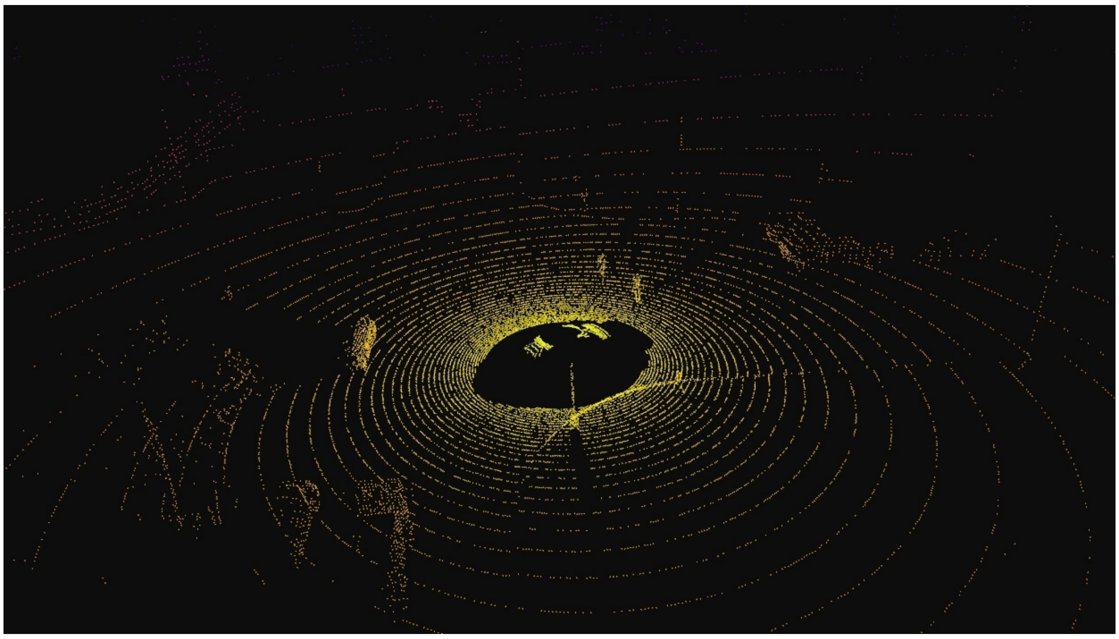


Figure 3.1: The resulting point cloud from the simulated lidar sensor in the CARLA Simulator

situations through the built-in scenario runner. Users can define scenarios with specific placement and actors' routes to support learning. In the annotation process, CARLA provides additional support, for example, bounding boxes of actors accessible through client scripts, just as classifications in semantic segmentation through a special camera-like sensor.

Altogether, the CARLA simulator is used in this work to generate LIDAR data in different traffic situations, annotated automatically with data extracted from the simulation through client scripts. In Figure 3.1, one such point cloud frame from CARLA is seen in combination with a virtual camera image. This circumvents the need to invest in a real-world LIDAR scanner, as especially models with a wide field of view are expensive. As this thesis aims to model some dangerous situations for cyclists, it further eliminates the need to collect data in the real world, and especially to put oneself in such a situation.

CHAPTER 4

Methodology

Our approach to the generation of the dataset is described in this chapter. We use the CARLA simulator to create a virtual world with virtual sensors and then simulate traffic behavior.

First, the general process is discussed to give an overview of the individual stage, and then each step is presented and detailed with our choices and reasons. Last, we describe the technical details of the final dataset.

4.1 Process

We structured our approach as a pipeline, where most stages are combined in a python script that accesses the local CARLA server. The script loosely follows the KITTI-CARLA dataset and data-generator [Des21]. The resulting pipeline is presented in Figure 4.1 and described in the following chapter.

4.2 Parameters

After the CARLA server is initially started, our client script sets general parameters about the whole simulation. They are general values that determine the type of data generated, like the *number of frames* per simulated map and the *frames per second* the engine should render.

The CARLA simulator is set to a fixed time-step, equal to 1 / *frames per second* when working with sensors because precision between sensors is relevant. Together with the activated synchronous mode, this enables the client script to process the data of each sensor while the server waits and does not overflow the client with information. In synchronous mode, the physics computation requires *maximum substeps* and a *maximum*

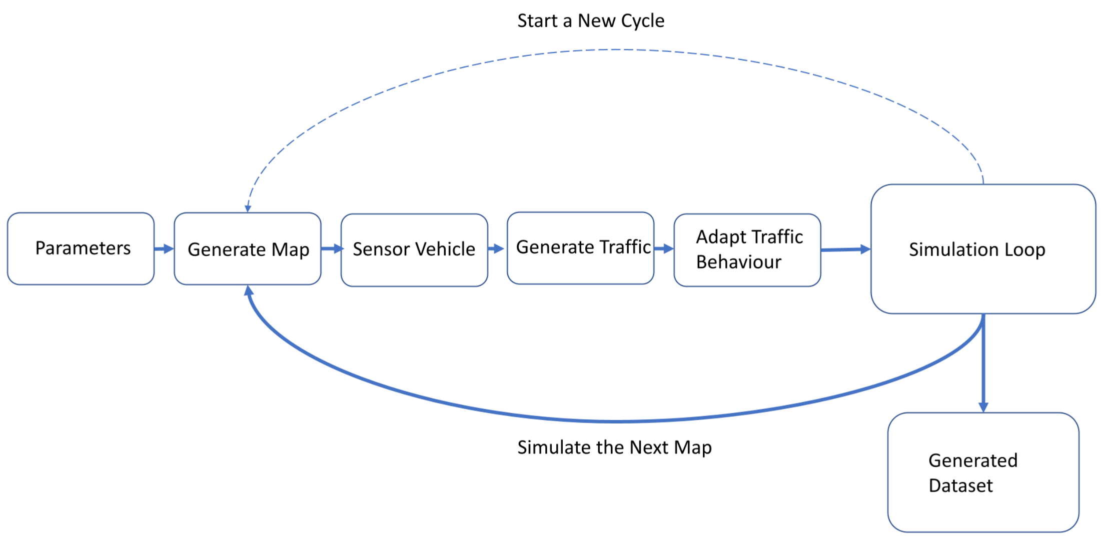


Figure 4.1: Pipeline describing the data generation process.

substep delta time to allow for precise sensor simulation. These are set per default to 10 and 0,01, respectively. Altogether, the condition

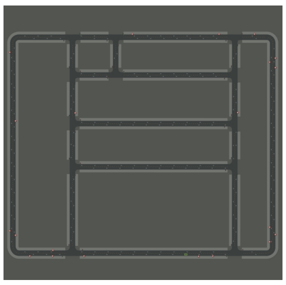
$$\frac{1}{frames\ per\ second} <= max\ substeps\ delta\ time * max\ substeps \tag{4.1}$$

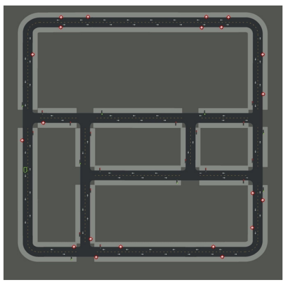
must be fulfilled. Therefore the *frames per second* are set to 10 throughout the simulation process.

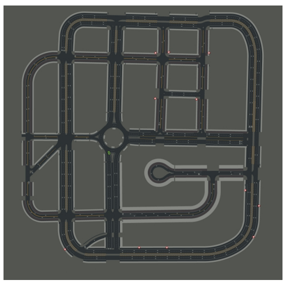
Additionally, the *sensor range* is set to 50 m, so the engine only collects specific sensor data if other vehicles are near the one recording.

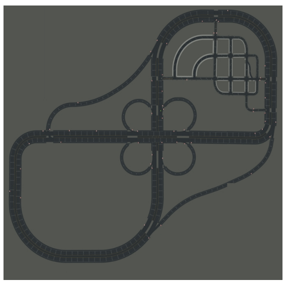
4.3 Generating the Map

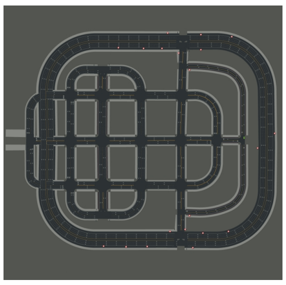
Once the client connects to the server, it can control the simulation. Now values that may differ for each simulation run are set, so various situations are generated at the end. As seen in Figure 4.2 CARLA offers 8 maps of streets and buildings, ranging from simple junctions in small towns to squared-grid cities and highways. For each run on each additional map, conditions such as the weather can vary, which is partly relevant for the LIDAR sensor. Different weather should generally affect LIDAR measurements because precipitation in the atmosphere interferes with the laser. In CARLA, the weather does not affect the physics simulation nor the LIDAR raycasting, but it is important to note in case future developments in CARLA make this possible.

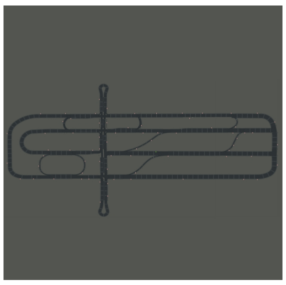

A basic town layout consisting of "T junctions".

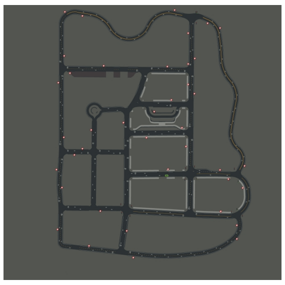

Similar to Town01, but smaller.


The most complex town, with a 5-lane junction, a roundabout, unevenness, a tunnel, and more.


An infinite loop with a highway and a small town.


Squared-grid town with cross junctions and a bridge. It has multiple lanes per direction. Useful to perform lane changes.


Long highways with many highway entrances and exits.


A rural environment with narrow roads, barns, and hardly any traffic lights.

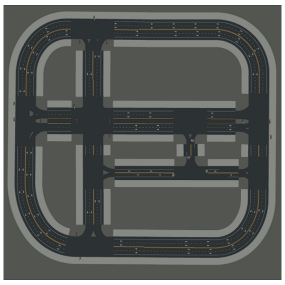

A city environment with different environments, such as an avenue or promenade, and more realistic textures.

Figure 4.2: The layout of the different maps available in the CARLA simulator.

4.4 The Sensor Vehicle

We chose a bicycle for the vehicle that carries the sensors and remains at the center of the simulation. We chose the cross bike from the CARLA library as the model and ego-vehicle to attach the sensors and measure the distance to other vehicles. The ego-vehicle is set to autopilot, so the simulations traffic manager handles its behavior, and it drives around automatically.

The LIDAR sensor is placed on the bicycle, so the driver does not obstruct the view of the surroundings. Its parameters were set closely to the Velodyne HDL-64E used in the KITTI project [GLU12, GLSU13], 64 *channels*, a *range* of 80 meters, a *rotation frequency* of 10 Hz, a 380° by 26.8° *field-of-view*, and 2 200 000 *points per second*. We also attached a GNSS sensor to map the local point cloud around the ego-vehicle into world space and put it into reference with other road users' trajectories. Additionally, the script enables attaching semantic segmentation, depth, and normal cameras to the ego-vehicle. These enable the generation of ground truth for further ML models, but are not used in this thesis, as the dataset generated is aimed to be a foundation for object tracking.

Additionally, we attached a GNSS sensor to the ego-vehicle. It provides a GNSS measurement consisting of its position in the form of metric values for altitude, latitude, and longitude.

Due to the fixed time step and the physics process's synchronous mode, each sensor's frame-by-frame data is synchronized by CARLA.

4.5 Generate Traffic

To simulate traffic, we generate vehicles and fit them with the necessary virtual equipment. Each actor is set to operate on autopilot, with minor adjustments described in Section 4.6. CARLA provides predefined spawn points on the roads. For each vehicle, one of those is selected at random for variety. In the case of overlapping locations, the engine handles this by not spawning colliding vehicles.

Each vehicle is fitted with a collision sensor. It registers as soon as the car collides with an object. Because of the strategy behind the autopilot, the only possible collisions with the ego-vehicle adhere to the rules defined in Section 4.6. If such a collision occurs, the vehicle responsible is removed, and the timestamp is marked. This way, the simulation can continue because the virtual bicycle remains intact.

4.6 Adapt Traffic Behaviour

The CARLA traffic manager provides each car with a simple autopilot with simplified behavior. This includes several patterns like the vehicle's target speed being fixed at a certain percentage of the current speed limit, cars are not goal-oriented and choose their path randomly at junctions. Furthermore, junction priority does not follow traffic regulations but rather an internal priority system. The CARLA API provides methods to influence this autopilot up to a certain point.

For the dataset generation, we applied some modifications to create situations the standard autopilot would not allow. One target is to enable the recording of situations where one vehicle might pose a threat to the bicycle or comes into close proximity. We randomly selected cars to be dangerous to the ego-vehicle. We achieve this by decreasing the *minimum distance* to the leading vehicle, changing the *relative speed difference* to the currently allowed speed limit, ignoring traffic lights and signs, and temporarily ignoring the ego-vehicle in trajectory calculation.

This increases the chance for such dangerous situations to occur without disturbing the flow of traffic too much. This could lead to undesired situations due to the limitations of the traffic manager.

4.7 Simulation Loop

Various sensor measurements are taken for each simulation tick, and vehicles are detected. As mentioned in Section 4.2, only certain situations are desired. At least one vehicle has to be inside a predefined minimum range of the ego-vehicle for LIDAR frames to be recorded. As soon as a vehicle enters or exits this radius, its relative position to the bicycle is calculated. Each sensor in the range is invoked to save data, or in the case of the collision detector, the car is removed if triggered.

During testing, the traffic manager showed certain limitations, as it came to a standstill, as seen in Figure 5.1. If the ego-vehicle is not moving for a certain amount of time, a new cycle in the simulation run is started. It starts under the same conditions defined in the stage described in Section 4.3, continuing with a reset of the spawned actors but with the remaining *number of frames* to be recorded in this map.

Once a map's desired quantity of frames is reached, the world is reset. The process starts again at the stage defined in Section 4.3 until all maps are simulated.

4.8 Dataset Specifications

The dataset is structured in data from the different sensors. We save each sensor output in a folder corresponding to each map. This work focuses on the data necessary for path tracing and object recognition through LIDAR measurements; therefore, these are at the center of the dataset. They serve as training data for such ML algorithms, while ground truth is provided through GNSS measurements representing the path a vehicle takes. Generally, the dataset is structured in a way that the data from each map is separated. Measurements of the respective runs are saved in subfolders of the map's name.

We set the *number of frames* per map to 20 000, with 10 *frames per second*. This results in more than 4 hours of continuous LIDAR coverage and 160 000 frames of LIDAR-generated point clouds over 8 maps.

Point Cloud

As mentioned in Section 4.4, the script supports the generation of video and depth camera data, but this work focuses on the data necessary for path tracing and object detection through LIDAR measurements. For each simulation tick, one LIDAR measurement is taken and saved. We use the provided method from the Python API, which stores the simulated point cloud in the *.ply* format. Therefore every frame is saved in the subfolder *frames* with the name being the corresponding frame number.

Location

The GNSS sensor attached to the ego-vehicle provides GPS measurements for each simulation tick to map the movement of the point cloud into the virtual space. Each location is saved in the *locations.csv* document, where the columns describe the altitude, latitude, and longitude, and the rows represent each distinct measurement.

Additionally, we use CARLA's internal coordinate system for location references of all vehicles. To describe the path of each nearby car in range of the ego-vehicle and the one of the ego-vehicle itself, we save their cartesian coordinates. A vehicle is considered nearby when it enters the *sensor range*. Therefore we describe the path of each car as a track of consecutive locations, each defined through an x, y, and z coordinate. All these locations are saved in a *trajectory_data.json* document, where the vehicle's id is the key, and each position's coordinates are saved in an array as values.

Evaluation Data

To evaluate our dataset, we measured the number of certain incidents and the distribution of passing distances in the *situations.json* file. A situation describes the initial direction of a car entering the *sensor range* of the ego-vehicle, the final direction, as well as the passing distance. The relative directions and passing distances are described in Chapter 5.

CHAPTER 5

Evaluation

The final dataset is generated to be used as an alternative or supplement to real world data for ML. This is to prevent overfitting on too small datasets and to include certain situations that are difficult or labor-intensive to recreate in the real world. The quantity of the generated data is adaptable by increasing the number of simulated frames. To ensure quality, we evaluate our work regarding the variety of situations observed.

Parameter	Value
Amount of cars	30
Amount of walkers	10
Distance to the leading vehicle	0,5 m - 1,5 m
Car's speed relative to the current speed limit	70% - 130%
Ego-vehicle's speed relative to the current speed limit	70%
Chance for a car to ignore traffic lights	20%
Chance for a car to ignore traffic signs	20%
Chance for a car to ignore the ego-vehicle up to a threshold distance	10%
Threshold distance from the ego-vehicle where cars may not ignore the ego-vehicle	1,5 m

Table 5.1: These are the behavioral parameters of the final simulation setup. We used the same values for each map and car, except if ranges are given. In these cases, The simulation chose random values inside the intervals to model diversity in car behavior. If not further specified, this setup applies to all examples displayed and tables given.

This chapter describes this evaluation process and how we adapted the simulation's behavior parameters. Unless specified otherwise, the values correspond to the ones described in Table 5.1. We left every other condition to the standard settings of the CARLA traffic manager. Each distance discussed in this chapter refers to the minimum distance between the bounding boxes of two CARLA Actors. It is calculated as the minimum of every euclidean distance between each vertex of both bounding boxes.

5.1 Results of Traffic Behaviour Adaptions

We also enforced and observed dangerous situations to utilize the capabilities of simulated situations compared to real-world situations. Our script utilizes the different possibilities to influence a vehicle's behavior offered by the CARLA API. The observed influences these adaptations had on simulation runs are discussed in this section.

Minimum Distance to Leading Vehicle

The CARLA traffic manager enables changing certain vehicles' *minimum distance* to other vehicles' front or back. In the simulator, we cannot change the distance to the side of vehicles. The autopilot uses this minimum value for all surrounding vehicles in the path calculation.

We wanted to enforce close proximities towards the ego-vehicle by lowering the *minimum distance* for some cars as soon as they enter the detection radius. Even though it leads to closer proximities of the ego-vehicle and certain vehicles, it also leads to undesired behavior. If the script selected cars already standing still, these would move closer toward their leading vehicle, resulting in unrealistic movement. Therefore we did not change the minimum distance during the simulation run, only set it once randomly between 0,5 m and 1,5 m to ensure variety observed by the sensors.

Ignorning Traffic lights and Signs

Cars can be manually set to ignore a percentage of all traffic lights. This results in these cars entering junctions even though other lanes would have priority. This only happens if the selected car is the first in line.

For our simulation, this generated situations where cars would drive at low speeds into intersections while other vehicles were currently inside. One such case is seen in Figure 5.1. The autopilot handles such events by slowing down each car involved, moving one at a time until the situation is resolved. This creates a temporary standstill for other vehicles. This behavior is not handled well by the traffic manager but creates situations where the trajectory of one car temporarily interjects with the ego-vehicle. For the final dataset, vehicles have a 5% chance of ignoring traffic lights and signs. This way, the ego-vehicle may observe such situations without the traffic flow being disturbed for too long.

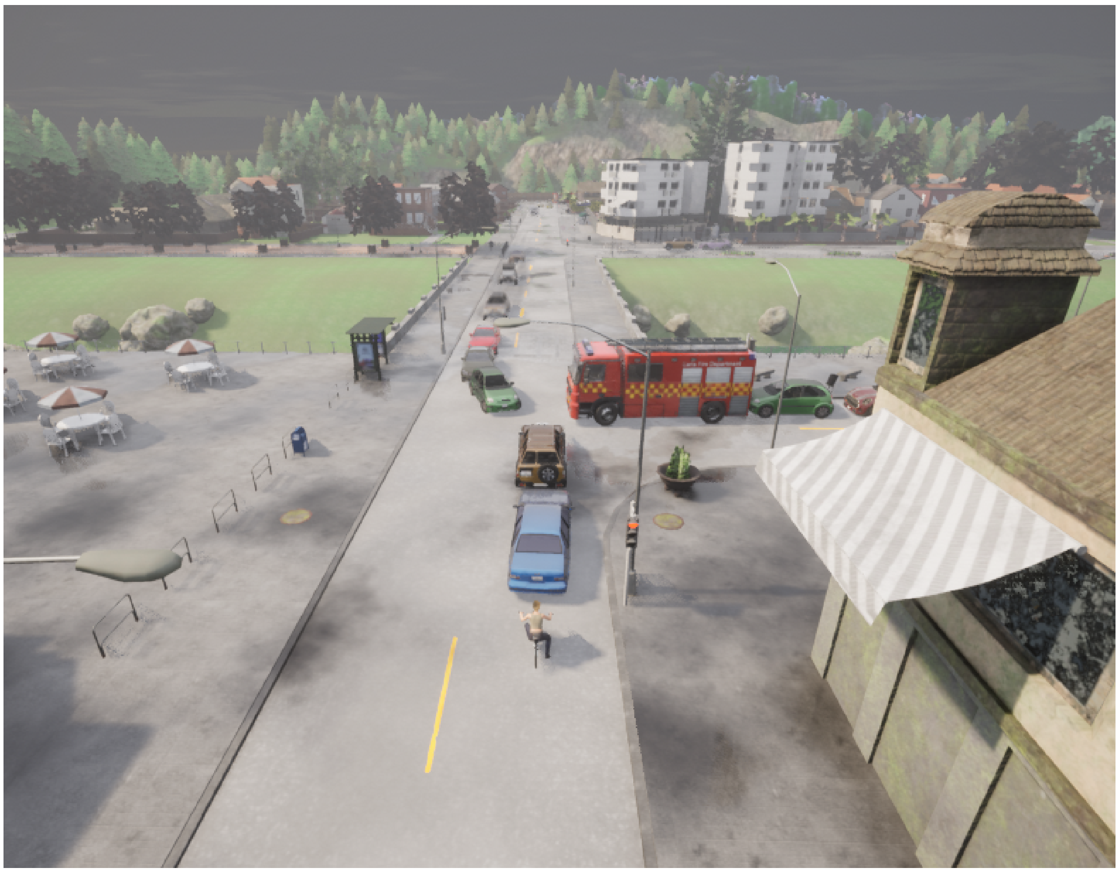


Figure 5.1: If the parameters of the cars regarding the ignorance of traffic signs and lights are set too high, a standstill occurs. This usually gets resolved through each car moving a small distance at a time but still brings the traffic to an almost total standstill.

Relative Speed Difference

For the final simulation run, we adapted each vehicle's target speed. To ensure variety, each vehicle's autopilot had different target speeds relative to the current speed limit. At the start of the simulation, this was set to a random value between 30% above and 30% below the speed limit for variety. We set the speed of the ego-vehicle to 70% of the current speed limit, to simulate the bicycle's slower speed compared to cars.

Ignoring the ego-vehicle

The CARLA API allows manually changing whether a specific vehicle ignores a percentage of all cars, one particular vehicle, or fully considers all of them. For our test simulations, we temporarily set cars to ignore the ego-vehicle until they reached a certain threshold distance from each other. This would result in crashes, where the car would be at full speed and could not break in time before colliding with the ego-vehicle. Therefore we

Chance for a car to ignore the ego-vehicle	Amount of collisions
5%	1
10%	2
15%	3
20%	3

Table 5.2: Amount of collisions between cars and the ego-vehicle, depending on the percentage of vehicles that ignore the ego-vehicle. These incidents were observed over 3 test runs of the simulation, in one map and 20 000 frames each.

limited the percentage of cars that may ignore the ego-vehicle. The frequency of collisions with these parameters is seen in Table 5.2, these were observed in simulation runs that do not represent the final dataset. As each collision with the ego-vehicle either disturbs traffic, or the simulation run is reset, we limited the chance of their appearance to 10%.

5.2 Variety of Situations

To ensure that our dataset is representative of the different kinds of movement in traffic, we collect data on car behavior during simulation. To measure the frequency of different types of relative positioning, we counted certain situations. As the LIDAR sensor's range is set to 50 m, one such situation starts as soon as a car enters this detection radius around the ego-vehicle. It ends once the car leaves this radius. This section covers the types of movement we evaluate and how they are categorized.

Relative Directions

To generalize movement from different sides of the ego-vehicle, we group the relative directions into right, behind, left, and front. This results in a 4x4 matrix where the rows represent the approaching direction and the columns the car's exit direction. Each cell describes how often such a situation was observed.

During the simulation, we measure this through our data generation script. Each time a car enters the detection radius, we save the initial location of the car and the ego-vehicle. Each time the car exits the detection radius, the car's final position is saved as well. The angle is calculated between the direction of the ego-vehicle and the relative positions of the ego-vehicle and the car. In Figure 5.2 the directions and angles of two cars concerning the ego-vehicle are shown as an overlay of an image taken from a birds-eye-view during a simulation run. The angle is calculated based on the *arctan2*, where the output is in the interval of [-180°; 180°]. The classification diagram is shown in Figure 5.3.

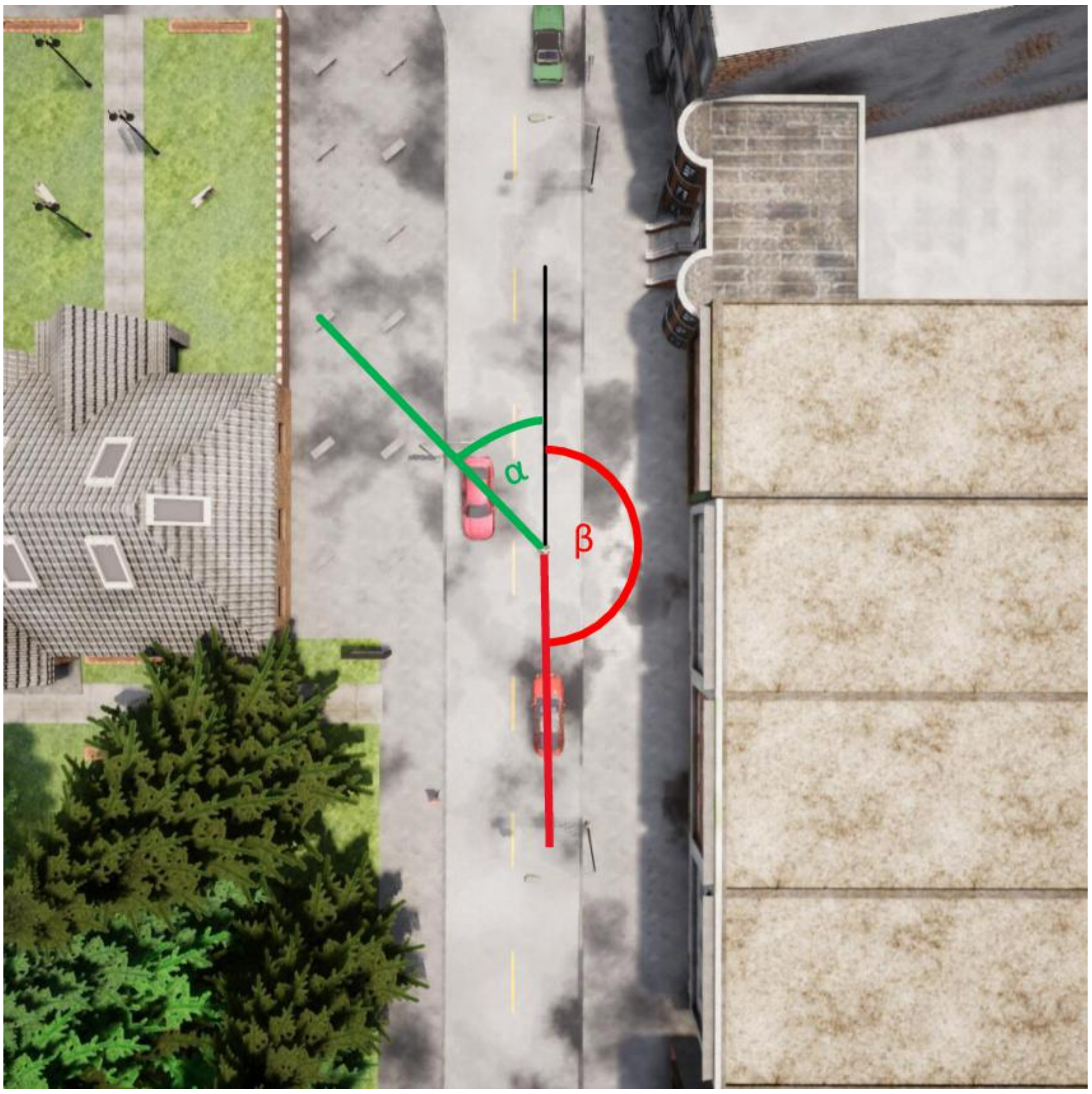


Figure 5.2: The angles α and β describe the current angle between the forward direction of the ego-vehicle, and the cars to the left and the right. In this instance, the bicycle representing the ego-vehicle is at the center and the angles are are 46° and 89° respectively. They would be classified as to the left and behind of the ego-vehicle.

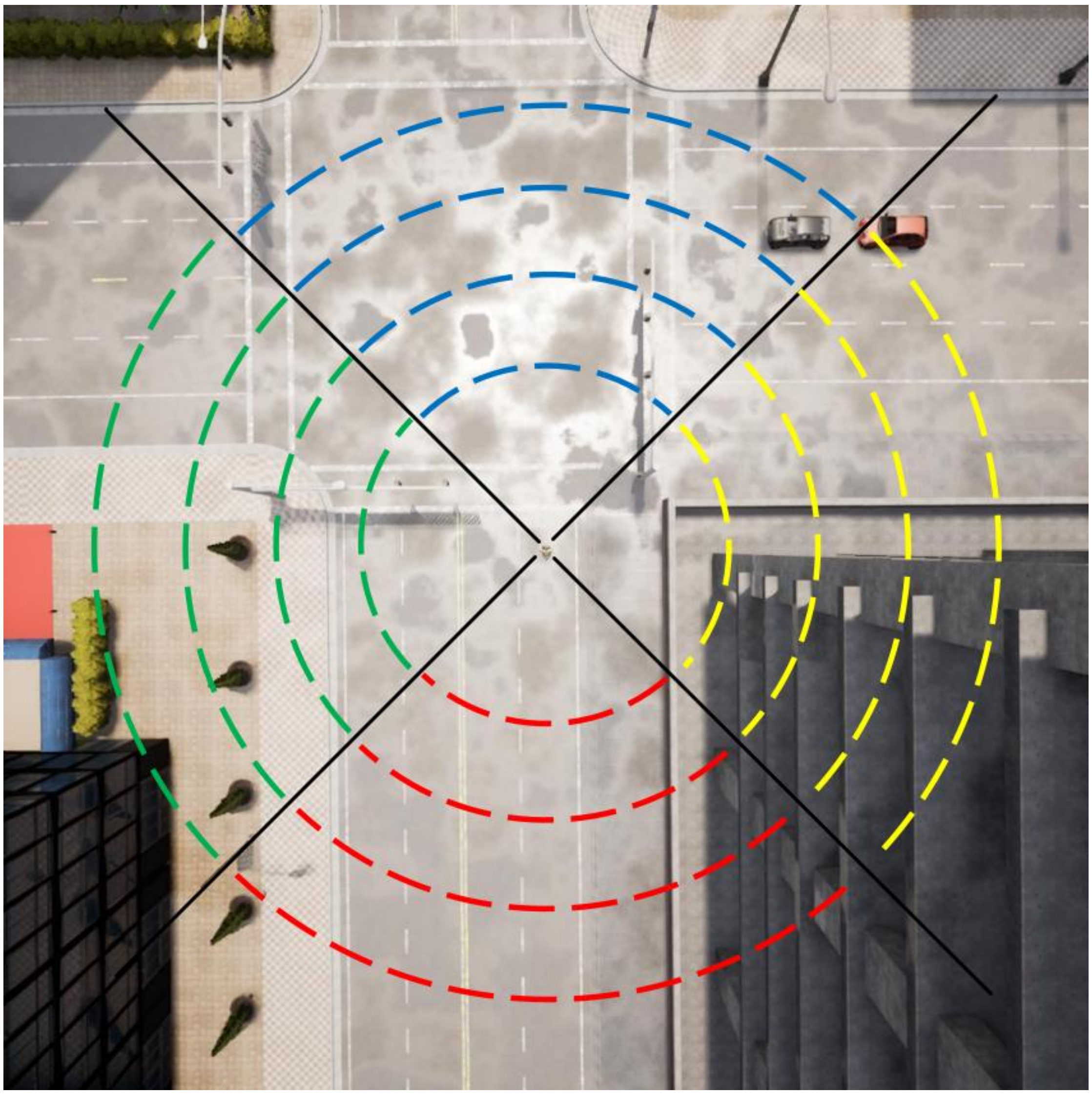


Figure 5.3: The four sectors around the vehicle cover a segment of 90° each. Each angle between the ego-vehicle direction and the vehicle is classified as either front, right, behind, or left.

initial \ final	*right*	*behind*	*left*	*front*
right	104	89	55	75
behind	136	139	166	114
left	65	68	41	53
front	130	127	114	96

Table 5.3: Amount of situations where a car approached the ego-vehicle from a certain initial direction left from a final direction. To sum up, 1 277 situatuins were observed.

Passing Distances

For our dataset to model all kinds of movement, it is necessary to record different proximities. For each situation observed, we measured the passing distance as the closest distance between the ego-vehicle and the car.

Speed of Passing Cars

We also evaluated our dataset regarding approaching cars and their relative speed towards the ego-vehicle. We were able to specify the distance to the leading vehicle through the CARLA API, and we also gave cars in a specific range around the bicycle a chance to ignore it up to 1,5 meters. This leads to some cars driving at full speed without consideration for the ego-vehicle, finally coming to a halt at the smaller radius. To evaluate the amount of such situations, we measured the closest passing distance and the distance between both actors 1 second before it was observed.

5.3 Results

For the final simulation run, the ego-vehicle was spawned into each of the 8 maps, together with 20 other vehicles. Over 20 000 frames per map, with 10 frames-per-second, this resulted in 16 000 seconds of observed traffic.

Each situation's initial and final angles were calculated and split into four categories. Table 5.3 shows how often each situation occurred. Generally, each type of movement, from each general direction towards each general direction is covered. There are differences between the number of times certain initial directions are covered. The most of the observed incidents are from the front or behind the ego-vehicle. This may be due to the ego-vehicle being generally slower than other vehicles, as it is left at the standard speed of 70% of the current speed limit and thereby slower than other traffic participants.

Figure 5.4 displays the passing distances observed in the final run. A variety of minimum distances from the ego-vehicle was observed. The peak at a close range of fewer than 5 m is due to cars moving in different directions on the same street. As seen in Figure 5.1, situations of close proximities inside junctions appear, however, CARLA does not feature a built-in way to differentiate between such cases.

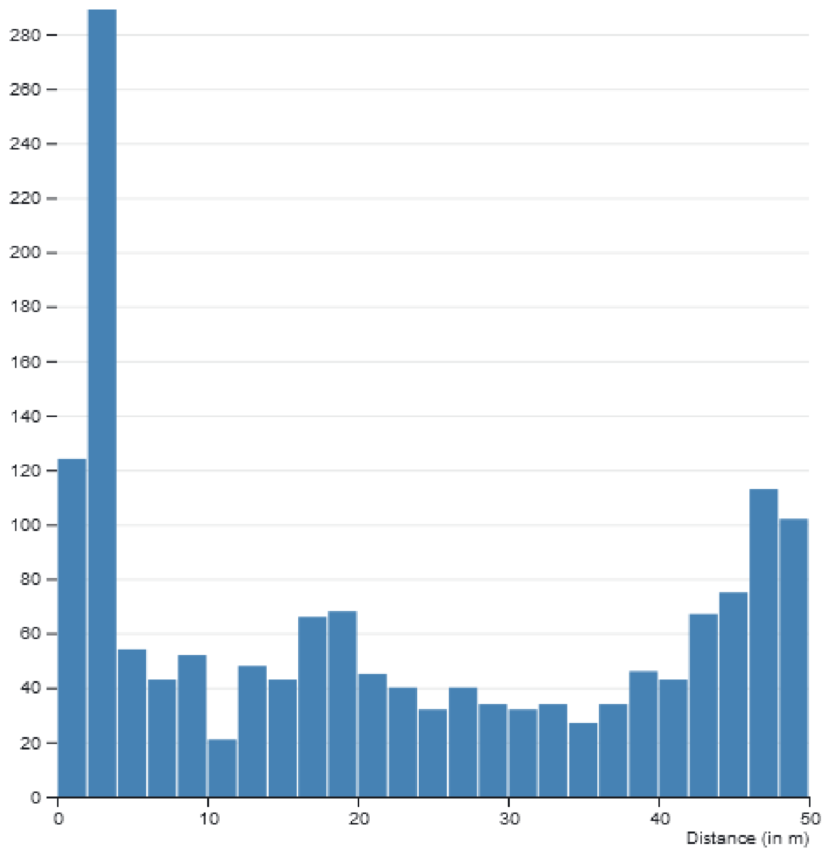


Figure 5.4: A histogram displaying how often cars pass the ego-vehicle at specific distances.

In Figure 5.5 we display how often combinations of minimum passing distances and distances 1 second before are observed. We aim to give insight into how we cover approaching vehicles and their relative speed to the ego-vehicle. The combinations cluster around closest distances between 2 and 2,5 meters along the horizontal axis. Along the vertical axis, we see a peak in a range from 2 to 8 meters of distance before 1 second. This is due to the lane width, where passing cars from opposing lanes are consequentially near. Another reason may be the extent of junctions and crossings. Cars waiting to enter are located certain distances away from the path of the bicycle if it crosses the intersection. The heatmap displays the internal controls of the CARLA traffic manager, especially the distance when it starts to slow down and consider other vehicles. Our adaptations regarding cars ignoring the bicycle are visible with the few outliers of far distances 1 second before the closest proximity was reached. In the row of closest distances of less than 0,5 meters our adaptations are visible as well. These situations mostly appear due to cars entering junctions from more than one lane at a time. Therefore they mainly appear due to the increase of the *chance for a car to ignore traffic lights*, as well as *chance*

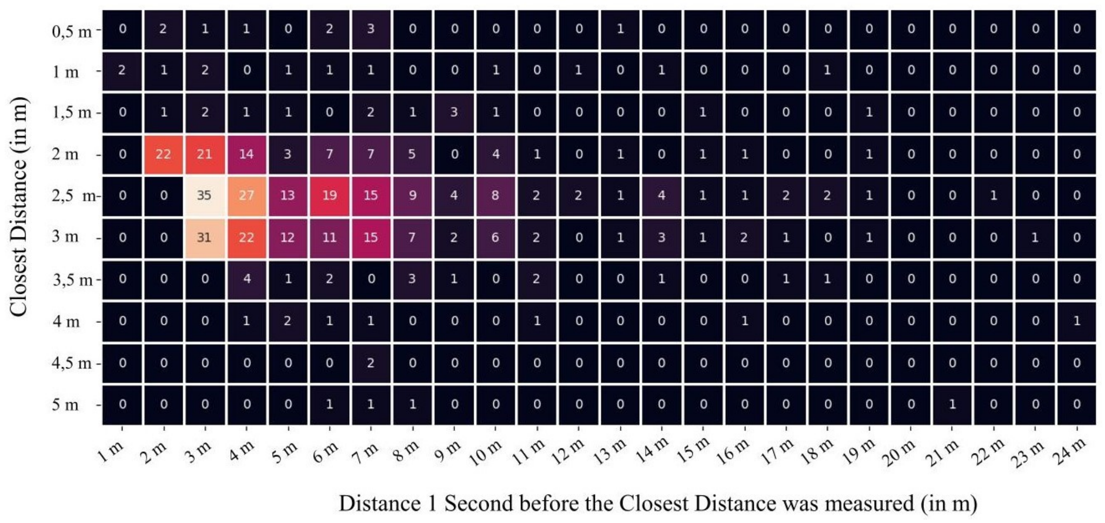


Figure 5.5: A heatmap displaying the correlation of the closest passing distance and the distance 1 second before it was measured. Each row represents the amount of situations in an interval of 0,5 meters up to the specified distance, and the columns have an interval size of 1 meter. For this sample of the dataset situations with a minimal passing distance less than or equal to 5 meters were considered.

for a car to ignore traffic signs. Fine-tuning of these parameters, especially the *Car's speed relative to the current speed limit* and *Chance for a car to ignore the ego-vehicle* will increase the chance of such situations to appear.

CHAPTER 6

Conclusion and Future Work

Our aim for this work was to create a point cloud-based dataset to model traffic scenarios that can be used as training data for ML algorithms. As we further specified the goal of capturing dangerous situations, complications arose. The cost of a LIDAR sensor capable of detecting in a 360° field-of-view combined with the human resources associated with collecting such data in real life exceeded the scope of this thesis. Therefore we chose the CARLA open-source simulator for autonomous driving research to generate this dataset in the virtual world.

This tool offers ready-to-use assets of traffic actors, like cars and bicycles, in different maps controlled by a traffic manager. The provided API can influence these actors to enforce certain situations. Regarding capturing dangerous situations, we modified relative vehicle speeds, compliance with traffic rules, and attention to other actors. CARLA also enables the generation of sensor data through virtual sensors, simulating LIDAR data, GPS measurements, and camera snapshots. We use these tools in a script to generate a dataset in 8 maps, with 160 000 frames. This resulted in a total of more than four hours of observed traffic.

We evaluated this dataset to determine how well it is suited to model real-world behavior. We also discuss the CARLA simulator's limitations regarding enforcing such behavior. In general, we showed how our dataset covers the movement of cars in every direction relative to an ego-vehicle, even though not all cases are observed equally often. Irregularities are due to the difference in speed between the actors. We are able to show how we can successfully influence the appearence of certain close-proximity situations, even though they do not happen very often. The traffic manager and autopilot provided by CARLA show their limitations regarding modeling natural traffic flow. In the case of dangerous behavior or close proximities to the ego-vehicle, traffic often comes to a standstill. To model the carelessness of drivers, we temporarily disabled their detection of the ego-vehicle. This resulted in total disregard for this vehicle and constant crashes during simulation runs. Therefore we limited such behavior in the generation of the final dataset to ensure steady traffic flow and undisturbed collection of data.

This work is intended to be used in the pipeline of ML as training data for tasks like object detection and path prediction. Related work suggests such synthetic data in combination with preexisting real-world data, to be used for such purposes. This ensures quality with realistic measurements while still benefitting from the quantity that can be generated as desired in a simulator.

Future work may improve the simulated data, especially regarding the modeling and distribution of desired situations. The parameters that can be set to increase the likelihood of close collisions can be fine-tuned further to allow for behavior between total ignorance and full consideration. The simulation would benefit from carefully selected triggers for certain car movements. The evaluation of such datasets may be improved by evaluating velocities and taking the directions of other vehicles into account, when classifying them into situations. CARLA also offers more complex scenarios via the CARLA Autonomous Driving Challenge, which were not explored in this work.

List of Figures

List of Tables

Bibliography

[AAMM$^+$18] Hassan Abu Alhaija, Siva Karthik Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *International journal of computer vision*, 126(9):961–972, 2018.

[BGM$^+$19] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9296–9306. IEEE, 2019.

[BVL19] Åsmund Brekke, Fredrik Vatsendvik, and Frank Lindseth. Multimodal 3d object detection from simulated pretraining. In *Communications in Computer and Information Science*, volume 1056 of *Communications in Computer and Information Science*, pages 102–113, Cham, 2019. Springer International Publishing.

[COR$^+$16] 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 *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223. IEEE, 2016.

[Des21] Jean-Emmanuel Deschaud. KITTI-CARLA: a KITTI-like dataset generated by CARLA Simulator. *arXiv e-prints*, 2021.

[DRC$^+$17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *CoRR*, 2017.

[Dun20] Nguyen Mau Dung. Super-Fast-Accurate-3D-Object-Detection-PyTorch. `https://github.com/maudzung/Super-Fast-Accurate-3D-Object-Detection`, 2020.

[FKG13] Jannik Fritsch, Tobias Kuhnl, and Andreas Geiger. A new performance measure and evaluation benchmark for road detection algorithms. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 1693–1700. IEEE, 2013.

[GHDEB$^+$20] Carlos Gómez-Huélamo, Javier Del Egido, Luis M. Bergasa, Rafael Barea, Elena López-Guillén, Felipe Arango, Javier Araluce, and Joaquín López. Train here, drive there: Simulating real-world use cases with fully-autonomous driving architecture in carla simulator. In *Advances in Intelligent Systems and Computing*, volume 1285 of *Advances in Intelligent Systems and Computing*, pages 44–59. Springer International Publishing, Cham, 2020.

[GLSU13] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The kitti dataset. *The International journal of robotics research*, 32(11):1231–1237, 2013.

[GLU12] 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. IEEE, 2012.

[GWCV16] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtualworlds as proxy for multi-object tracking analysis. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4340–4349. IEEE, 2016.

[HWC$^+$20] Xinyu Huang, Peng Wang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. The apolloscape open dataset for autonomous driving and its application. *IEEE transactions on pattern analysis and machine intelligence*, 42(10):2702–2719, 2020.

[JRBM$^+$16] Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? *arXiv preprint arXiv:1610.01983*, 2016.

[KPA$^+$22] Andreas Kloukiniotis, Andreas Papandreou, Christos Anagnostopoulos, Aris Lalos, Petros Kapsalas, D.-V. Nguyen, and Konstantinos Moustakas. Carlascenes: A synthetic dataset for odometry in autonomous driving. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 4519–4527. IEEE, 2022.

[LXG21] Yiyi Liao, Jun Xie, and Andreas Geiger. KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *arXiv preprint arXiv:2109.13410*, 2021.

[MG15] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[Nik21] Sergey I. Nikolenko. Synthetic simulated environments. In *Synthetic Data for Deep Learning*, Springer Optimization and Its Applications, pages 195–215. Springer International Publishing, Cham, 2021.

[NOBK17] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulo, and Peter Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5000–5009. IEEE, 2017.

[OPR23] Stefan Ohrhallinger, Simon Pointner, and Michael Rubik. Cyclesafely. `https://gitlab.cg.tuwien.ac.at/stef/cyclesafely/-/blob/main/carla-simulator`, 2023.

[ROP23] Michael Rubik, Stefan Ohrhallinger, and Simon Pointner. Cyclesafely, data generation script. `https://gitlab.cg.tuwien.ac.at/stef/cyclesafely/-/blob/main/carla-simulator/kitti-carla/datagen_main.py`, 2023.

[SL21] Weisong Shi and Liangkai Liu. *Computing Systems for Autonomous Driving*. Springer International Publishing Imprint: Springer, Cham, 1st ed. 2021. edition, 2021.

[SS02] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.

[WWY$^+$21] Patrick Wenzel, Rui Wang, Nan Yang, Qing Cheng, Qadeer Khan, Lukas von Stumberg, Niclas Zeller, and Daniel Cremers. 4seasons: A cross-season dataset for multi-weather slam in autonomous driving. In *Pattern Recognition*, Lecture Notes in Computer Science, pages 404–417. Springer International Publishing, Cham, 2021.

[YDB$^+$18] Han Woong Yoo, Norbert Druml, David Brunner, Christian Schwarzl, Thomas Thurner, Marcus Hennecke, and Georg Schitter. Mems-based lidar for autonomous driving. *Elektrotechnik und Informationstechnik*, 135(6):408–415, 2018.