AGH University of Science and Technology

FIELD OF SCIENCE: Engineering and Technology

SCIENTIFIC DISCIPLINE: Mechanical Engineering

DOCTORAL THESIS

Improvement of Advanced Driving Assistant Systems performance with the application of high quality synthetic data

Author: mgr inż. Paweł Filip Jabłoński

First supervisor: dr hab. inż. Joanna Iwaniec, prof. AGH
Assisting supervisor: mgr inż. Tomasz Powroźnik

Completed in: AGH University of Science and Technology, Faculty of Mechanical Engineering and Robotics, Department of Robotics and Mechatronics

Krakow, 2023

Promotorowi pracy
Pani dr hab. inż. Joannie Iwaniec, prof. AGH składam serdeczne podziękowania za pomoc, poświęcony czas oraz wkład merytoryczny w trakcie realizacji tej pracy.

Dziękuje mojej Żonie Magdalenie za wytrwałość i codzienne wsparcie, dzięki któremu ta praca mogła powstać.

Dziękuje mojemu Bratu Michałowi za poświęcony czas i pomoc przy realizacji publikacji oraz Panu dr inż. Danielowi Prusakowi, dzięki któremu rozpocząłem moje studia doktoranckie.

Contents

List of Acronyms

ADAS	Advance Driving assistant systems
SAE	Society of Automotive Engineers
MiL	Model in the loop
SiL	Software in the loop
HiL	Hardware in the loop
VEHiL	Vehicle in the loop
CLHiL	Hardware in the closed loop
AEB	Automatic Emergency Breaking
ACC	Adaptive Cruse Control
TSR	Traffic Sign Recognition
LKA	Lane Keeping Algorithm
LDW	Lane Departure Warning
FSD	Free Space Detection
Carla	Car Learn To Act
YOLO	You Only Look Once
TP	True positive
FP	False positive
FN	False negative
TN	True negative
UE4	Unreal Engine 4
GTA V	Grand theft Auto 5
AP	Average Precision
mAP	Mean average Precision, calculated for all classes
WSL	Windows subsystem for Linux
RAM	Random access memory
ME	MobilEye
FDX	Fast Data Exchange protocol
Y	Lateral distance
X	Longitudinal distance
C_r	Curvature rate
C	Curvature
C_0	Lane position at x(0) [m]
H	Heading angle
L_c	Lane center
$C_{0\Delta}$	Lane offset from the center at x = 0 [m]
Y_r,Y_l	lateral distance to the left and right destination point
X_d	Longitudinal distance to destination point
A, B, C, D	Parameters of circle function
r	radius of the circle
α	steering wheel angle
l	distance between vehicle axles
BEV	Birds eye view

P_{ps}	LiDAR points generated per second
FOV_h	Horizontal field of view
N	Number of LiDAR sensor channels
f	Rotation frequency
ϕ_h	Horizontal angular resolution
Ψ	Azimuth angle
θ	Elevation angle
r_s	Radius
I/I_0	Point intensity
a	Atmosphere attenuation rate
CNNs	Convolution neural networks
IoU	Intersection over Union
p_r	Precision
r_c	Recall
g	Grayscale value of 8 bit pixel
R_1, G_2, B_3	R_1 red , G_2 green and B_3 blue color channel of image pixels
SSL	Semantic segmentation LiDAR

Streszczenie

Niniejsza praca doktorska poświęcona została badaniom użyteczności oraz możliwości zastosowania danych syntetycznych w zaawansowanych systemach wspomagania kierowcy. Tego typu systemy percepcji wykorzystują czujniki takie jak kamera, radar czy LiDAR. Dzięki rozwojowi technologii uczenia maszynowego, a w szczególności uczenia nadzorowanego głębokich sieci neuronowych, systemy percepcji zaczęły osiągać wysokie poziomy wydajności. Tego typu algorytmy wizyjne detekcji obiektów wymagają zaawansowanych testów walidacyjnych zanim zostaną wdrożone i dopuszczone do ruchu publicznego. Najnowsze osiągnięcia technologii renderowania 3D, takie jak UE czy Unity, umożliwiają generowanie realistycznych danych syntetycznych.

W pierwszych rozdziałach pracy zawarto wprowadzenie w zagadnienia związane z systemami wspomagania kierowcy, nakreślono zakres przeprowadzonych prac badawczych, sformułowano tezę główną i tezy pomocnicze, a także przedstawiono krytyczny przegląd aktualnego stanu wiedzy.

W rozdziale czwartym przedstawiono eksperymenty z użyciem generowanych obrazów ze scenariuszy drogowych, które zostały użyte do walidacji pracy algorytmów detekcji znaków drogowych oraz utrzymania pasa ruchu. We wszystkich opisanych eksperymentach zastosowane zostało oprogramowania o nazwie CARLA. Jest to symulator wykorzystujący technologię UE, pozwalający generować w czasie rzeczywistym obrazy dla takich czujników jak kamera, LiDAR czy radar. Badania zostały przeprowadzone z dużym naciskiem na możliwość pracy systemu walidacyjnego w czasie rzeczywistym, a ich celem było zweryfikowanie użyteczności danych syntetycznych do walidacji pracy zaawansowanych systemów wspomagania kierowcy. Takie podejście pozwoliło opisać i z sukcesem przeprowadzić test pracy algorytmu utrzymania pasu ruchu z zastosowaniem kamery MobilEye 6 w pełnej pętli sprzężenia zwrotnego. Jest to pierwszy tego typu opisany test algorytmu utrzymania pasa ruchu wykorzystującego dane syntetycznie generowane w czasie rzeczywistym przez symulacje, która reaguje na informacje z czujnika.

W najnowszych systemach wspomagania kierowcy wykorzystywane są nie tylko kamery, ale również radary oraz LiDARy. LiDARy wielokanałowe są nową i mocno rozwijającą się technologią. W rozdziale piątym przedstawiono eksperymenty dotyczące użyteczność syntetycznie generowanych danych czujników LiDARowych w aspekcie algorytmów wspomagania kierowcy. Do generacji takich danych wykorzystano technologię rzutowania promieni, która wspierana jest przez symulator CARLA. W eksperymencie przedstawiono przykład wykorzystujący syntetycznie wygenerowaną chmurę punktów jako dane wejściowe do algorytmu wykrywania przestrzeni wolnej przed pojazdem. Uzyskane wyniki zostały porównane z wynikami otrzymanymi dla pojazdu w warunkach drogowych.

W rozdziale szóstym przedstawiono badania wykorzystujące syntetyczne dane do wspomagania uczenia maszynowego, w szczególności głębokich sieci konwolucyjnych. Algorytmy uczenia nadzorowanego wymagają zastosowania bardzo dużych ilości danych uczących, w celu uzyskania zadowalającego poziomu skuteczności. Takie dane nie tylko trzeba zgromadzić, ale również oznaczyć, aby mogły być użyteczne. Ten problem można ograniczyć poprzez zastosowanie automatycznie generowanych danych syntetycznych. W rozdziale 6 przedstawiono eksperymenty oraz metody pozyskiwania takich danych uczących dla detektora pieszych. Celem badania było sprawdzenie, czy dane syntetyczne mogą zastąpić dane rzeczywiste podczas uczenia algorytmu detekcji pieszych. Przedstawione eksperymenty porównują precyzję oraz dokładność działania detektora uczonego z wykorzystanie syntetycznych, mieszanych oraz rzeczywistych danych. W

procesie generacji syntetycznych danych wykorzystano autorską metodę precyzyjnego oznaczania pieszych na danych z czujnika LiDAR, która wykorzystuje dane segmentacyjne symulacji. W procesie uczenia detektora zastosowane zostały dane przetworzone poprzez projekcję trójwymiarowej chmury punktów na dwuwymiarową mapę głębokości. Wskaźniki jakości działania detektora pieszych zostały obliczone z użyciem tylko rzeczywistych danych pochodzących z LiDARu. Jako zbiór danych rzeczywistych w eksperymentach wykorzystano zbiór Waymo open dataset. Opisane w pracy wyniki potwierdziły, że tak wygenerowane dane syntetyczne nie tylko mogą być wykorzystane do uczenia maszynowego, ale również poprawiają ogólną wydajność algorytmu detekcji pieszych. Użycie automatycznie generowanych danych syntetycznych pozwala skrócić czas generowania zbioru danych do uczenia algorytmów sieci głębokich, jak również może poprawić ich ogólną wydajność.

Summary

This doctoral thesis concerns investigation into synthetic data applications and usability in the advanced driving assistant systems. This type of perception systems is based on camera, radar or LiDAR sensors. With the development of machine learning technology, especially supervised deep neural network learning, perception systems have begun to achieve high levels of performance. That type of vision algorithms requires advanced testing and validation before it may be used on the public roads. The newest rendering engines, such as UE or Unity, allow to generate realistic synthetic data.

The first chapters of this thesis include description and characteristic of advanced driving assistant systems and the range of conducted experiments. Furthermore, there is the main and auxiliary thesis formulated, together with the critical overview of the actual state of the knowledge in the field.

Research, where synthetically generated data was used for validation of sign and line detection algorithms, is described in the 4th chapter of this thesis. In the course of the conducted experiments the CARLA software was used. It is a simulator, which bases on the UE technology and allows for real-time generation of images out of cameras and other type of data from radars or LiDARs. The research was carried out with a strong emphasis on the possibility of the real-time operation of the validation system. It aimed at verifying the usefulness of synthetic data for validation of the operation of advanced driver assistance systems. This approach made it possible to successfully conduct the lane keeping algorithm test for MobilEye 6 camera sensor in the full closed feedback loop. This is the first described test of a lane keeping algorithm using the data synthetically generated in real time by a simulation that responds to the sensor information.

The newest driving assistant systems relay not only on cameras but also other types of sensors like radar or LiDAR. Multi-channel LiDARs are the new type of sensors in the automotive field. In the 5th chapter of this thesis, the usability of synthetic point cloud data of LiDAR sensor is experimentally verified. To be able to generate LiDAR sensor point cloud, the ray-casting technology was used. This technology is supported by CARLA simulator. Experiment described in the chapter 5 shows the example usage of synthetic point cloud as an input data for the free space detection algorithm. Experiment results were compared to the results acquired by the real sensor on the public road.

Chapter 6 of this work presents research where synthetic data was used to support the machine learning, specifically training of convolutional neural networks. Supervised learning algorithms require very large amounts of training data to achieve the satisfactory levels of efficiency. Such data not only has to be collected, but also labeled in order to be useful. This problem can be reduced through the use of automatically generated synthetic data. Chapter 6 presents experiments and methods for obtaining such training data for a pedestrian detector. The purpose of the study was to demonstrate whether synthetic data can replace real data when training a pedestrian detection algorithm. The presented experiments compare the performance of the detector operation using synthetic, mixed, and real data to train the model. An proprietary method for precisely labeling pedestrians in LiDAR sensor data using segmentation data from simulation was used to generate automatically labeled training data. The detector was trained on the prepossessed data. Processing included a three-dimensional point cloud projection onto a two-dimensional depth map. The performance indicators of the pedestrian detector were calculated using real LiDAR sensor data only.

In the experiments, the Waymo open dataset was used as the real data set. The results described in the work show that such synthetic data can not only be used for machine learning, but it can also improve the overall performance of the pedestrian detection algorithm. The usage of automatically generated synthetic data can shorten the time to generate a data set for training algorithms and may also improve its overall performance.

1 Introduction

Nowadays, the goal of automotive industry is clear: to build a fully autonomous vehicle. Such vehicle allows to travel from point a to b without the necessity of having a driver on board. Researchers in automotive industry develop a lot of new advanced systems in vehicles, which makes it possible to achieve the higher and higher levels of autonomy. That type of systems, supported by the electronics and sensors, is called advanced driving assistant systems (ADAS). The levels of autonomy are defined by the Society of Automotive Engineers (SAE) in the standard J3016 [1].The most advanced prototypes developed today are on the level 3 out of 5. Promising results achieved out of the new technologies like deep machine learning supported by high quality sensors with high computational capabilities of electronics, allow to think about autonomous driving in terms of when and not whether it is going to be applied. Nevertheless, to elevate the vehicle systems into the next levels of autonomous driving, the huge amount of work and money has to be invested into the development and research. What's even more challenging, the complexity and sophistication of driving system vastly increase on every level of automation. Good example of complexity of such systems, which are still under development, is the application of Waymo company [2]. Their vehicle is equipped with ten sensors, five cameras and five LiDARs fixed around the body of the vehicle. Image of Waymo vehicle is presented in figure 1.

Figure 1: Waymo company vehicle equipped with five high quality cameras and five lidars.

The highest interest is put into the machine learning technologies, due to technical possibilities and high performance available out of this technology. Machine learning as a core technology is highly valuable in terms of autonomous vehicle systems. It enables to deal with the main challenge, which is monitoring of the road environment. Properly trained deep neural networks provides highly reliable information about the road objects. One of the types of neural network training is called the supervised learning. Labeled training data is used as an input to deliver the necessary information about objects of interest, which allows to train the model. In case of the supervised learning, the amount and quality of labeled training data are of key importance. This method requires huge training datasets to reach high performance. Usually, the bigger dataset results in better results. Of course, if all data samples are diverse. The same input leads quickly to the problem called overfitting. Size of the training dataset is a first major challenge in this technique. A lot of effort has to be put to record huge amount of data and then, what's even more time consuming, to label the objects of interest on each sample. There is another issue related to

the data generation. Due to the rapid changes in the newly developed sensors, in most cases, the data recorded and labeled for old versions of the same sensor is useless in terms of the supervised learning. Neural network models are very sensitive to the changes in the input, which mostly lead to the poor performance of such a model. The combination of both mentioned issues leads to the situation where huge amount of work has to be put into each task to train a model good enough to be useful in real applications.

In the field of driving automation the researchers need big budgets from governments, corporations or both to build a prototype vehicle, record and label valuable data before they can experiment with machine learning models to achieve the best possible performance [3, 4, 5]. Usually, this expensive process of data preparation has to be repeated, if any sensor used for data recording is changed. For example the LiDAR sensor used for creating the nuScenes dataset had 32 channels, while the Waymo vehicle was equipped with 64 channel LiDAR. This differences make the LiDAR sensor data for nuScenes useless in the Waymo application and vice versa. To overcome this problem, the best solution would be to generate automatically more data, which would be useful for the supervised learning models. High performance computers may be used as the source of artificial data. The idea to generate synthetic data out of the computer simulations is not new, but the possibilities to achieve successful results are new indeed. One of the examples of usage of artificial images for ADAS validation was shown in [6], the figures depict how low was the quality of the rendered image. Computer graphic engines were not able to generate images comparable to any real road camera. Neural network models are sensitive to any changes in input, that is why the quality and character of synthetic data must be as close to the real data as possible. Modern computer simulations render high quality 3D environments [7, 8, 9], which makes it possible to use the synthetically generated data for the purposes of machine learning or any kind of perception system validation. Many attempts [9, 10, 11] on synthetic data usage lead the researchers to the phenomenon called *synthetic to real gap*, which is the gap between performance of the real data model and synthetic data model. In case of camera images it was found that the diversity and size of the synthetic dataset are the major factors [9] enabling to bridge the synthetic to real gap. The similar issue can appear if ADAS is validated on synthetically generated data. Poor quality or different character of data decreases the chance for the system to correctly detect the object, which, in turn, leads to the wrong test results. Due to this type of false negative results, many developers of ADAS does not work on issues related to their systems if those problems were found with the usage of artificial or synthetic data. That's why it is so important to prove that synthetic data has a very low or even no influence on the model performance in comparison to the real data usage. Proper synthetic data, which allows to validate ADAS, can significantly decrease the expenses and work load while new systems are developed.

There are multiple levels on which ADAS can be tested. The V-diagram represents the sequential design and validation phases of the critical safety systems. System validation methods are generally called X-in-the-loop, where X is the other type of the test setup. It starts from the MIL (model in the loop) and ends on the VEHIL (vehicle in the loop). Testing loops are also divided with respect to the type of data flow. The simple ones, where the input data from recording is injected into the system to verify if the output is correct, are called open loops. More complex methods, where the input data reacts on the test object output, are called closed loops. Hardware in the loop (HIL) with the closed loop method requires simulation or model, which is able to correctly react on the system output. The simple example of such a test is a setup which validates the AEB (automatic emergency braking function), where a vehicle has to stop before it hits the object on the

road. In case of the open loop testing, there are many applications that allow to test some ADAS functionalities [12, 13, 14]. The solution with the closed loop validation for ADAS is much more complex to deliver. That's why there are only a few examples of successfully developed closed loop environments for ADAS validation [15, 16]. Most of this examples are on VEHIL level of validation. As already mentioned, the quality of data deliver by simulation is of key importance for correct results of testing. By looking into the example images, which are rendered for the purposes of testing in mentioned publications, it is clear that the gap between real and synthetic data is huge. That is why there is a room for experiments with the same methods but on modern state of the art rendering engines. There is also one important topic concerning any X-in-the-loop tests. More complex and sophisticated validation methods require the real-time simulations and work of the whole test environment, which means that the whole data flow inside the environment has to be deterministic and repeatable. Otherwise the results may fluctuate and overall no valid information can be gathered out of the test system. The combination of the real-time and high quality rendering abilities is the key to deliver state of the art reliable and low cost results for ADAS development projects.

The perception part of the driving sensors is crucial. In many research projects, experiments with ADAS functionalities are run on the camera sensor, which is low cost and most commonly used sensor for such applications. Camera has a great performance is case of data resolution and its one of the cheapest type of sensor [17]. Due to this properties it is used as a core of most detection systems, but it is not the only sensor that can be used for such an application. There are also ultrasonic sensors, radars and LiDARs, and each of them has its own unique features. For example ultrasonic sensors has low abilities in case of driving automation, due to its low detection range. That is why the ultrasonic sensors are mainly used for parking maneuvers. Radars have much higher range of work than cameras and are not sensitive to the light condition. But their performance in ADAS applications is low due to the low accuracy and resolution. The big problem is also the mutual interference of the radar sensors. If any other radar is in the field of view of other radar, most likely both sensors starts to produce a lot of interfered data. Due to this problem, radars are mostly used for adaptive cruise control – in ACC or AEB applications, where only a small front field of view is necessary to be scanned. The work of LiDARs is based on light impulses that are used for the purposes of scanning the road environment. The biggest advantage of LiDAR over radar or camera is high accuracy and precision. LiDAR can replicate its surrounding in hundreds of meters range with just a few centimeters of accuracy, delivering spatial data about road environment. Rotary LiDAR field of view is 360 degrees, so it scans environment in all directions. High precision and accuracy, as well as 3D interpretation of the sensor surrounding are the features that make the LiDAR very useful sensor. There is one major problem with LiDARs: the cost. Right now the cost of each high performance LiDAR is so high that it cannot be applied in any production vehicle. That is why LiDARs are used only in the development projects and in some of the research projects. The precise ground truth information gathered from LiDAR sensors is very useful, for example in stereoscopic camera application, where the distance to the object has to be estimated. Due to highly valuable data gathered by LiDAR, the manufacturers put a lot of work to decrees the cost of sensor and additionally increase the performance, since nowadays multi-channel LiDAR is relatively new piece of hardware. Mentioned trend in LiDAR technology allows, for example, Waymo company [2] to equip their cars with five LiDARs. Popular manufacturer Velodyne [18] started with 8/16 channel LiDARs. Now there are sensors which provide high density point clouds with 32/64 or 128 channels. Valuable data and perspective of the further cost reduction, make

the LiDARs more and more usable in many of the robotic or automation application. Due to the reasons mentioned above and the fact that the multi-channel LiDARs are novel, many experiments presented in this work focus on this type of sensors.

State of the art computer simulations are able to create not only the RGB images but also 3D point clouds out of the simulation scenarios. Technology called ray-casting enables researchers to experiment with synthetically generated LiDAR sensor data. Similarly as for cameras this data can significantly decrease the expenses and time spent on the development of new ADAS functionalities supported by LiDAR sensors. Moreover, in case of really rare or even very dangerous situation, synthetic data can be the only way to do it. One of good example is the situation when a child is trying to cross the multi-lane highway. Any attempt on recording such a road incident would be very risky, but in high quality 3D rendering engine it's possible to record hundreds of frames of such scenario, without the risk that anyone gets injured. This type of data can be helpful in training or testing of ADAS or systems of other type. What's also important, all work can be done without going on the road and event without any physical hardware. Highly sophisticated and complex road simulation can substitute the real data.

2 Scope of work and thesis

The goal of this work is to experiment with the novel computer simulation methods to build robust, reliable and repeatable environment, which enables high fidelity validation and improves development of the advanced driving assistant systems for automotive industry. The main interest of the research was put on the feasibility study of 3D rendering capabilities of the modern graphical engines. Thanks to the computer game industry, road scenarios with all the traffic simulations are now available also for the purposes of development of automotive systems, which leads us to the autonomous driving. Increasing complexity of the vehicle ADAS generates more and more expenses and requires hundreds of hours of driving to develop and validate the new methods or technologies. There is an increasing demand to support this development by transferring the part of the work onto the computer simulations. The ability to generate synthetic data out of the simulation can not only decrease the expenses but might also improve the performance of the system. It might be obtained by more complex validation or by introducing the improvements of the training datasets prepared for the supervised learning.

The first point of the carried out research was to select the best available computer simulation software for the further experiments. The review of the available novel possibilities in the field is described, with the explanation why project called *Car learn to act - CARLA* was chosen [19].

Hardware in the loop (HIL) environment was developed, to experiment with synthetic data. HIL environment was used to validate ADAS functionalities, which are based on the camera and LiDAR sensors. The main focus was put on the real-time abilities of the HIL, to fulfill the requirements of the work thesis. Automation program called Morphee [20] was used as the core part of the developed test environment. Morphee is the first testbed for ADAS, which combines flexibility of applications on Windows operating system and the real-time capabilities on the same machine. Figure 2 depicts the four different stages of ADAS HIL development. In the considered figure there are presented setups for three different hardware cameras and two LiDAR sensors simulations.

Figure 2: Four different stages of the ADAS HIL development.

Research includes the experiments with the real ADAS sensor, Mobileye 660. Experiment results prove that the developed HIL is able to validate the traffic signs recognizing (TSR) func-

tionality run on the camera sensor. In the course of the carried out experiments the main stress was laid on determining the reliability and accuracy of the proposed method of validation. Based on the same camera sensor, the further experiments with closed loop validation on the HIL, were performed. The lane keeping algorithm (LKA) based on the line detection out of synthetic data was developed. Complex real-time closed looped experiment with Carla simulator and Morphee automation environment was the first described LKA functionality test, successfully evaluated with the usage of the Mobileye 6 camera sensor and the real-time synthetically generated data.

Algorithms for live processing of raw synthetic data into real-time LiDAR and camera sensor stream were developed. Algorithms were written in c# and c++ programming languages. Those algorithms were used to experiment with substitution of real hardware sensor to validate one of the ADAS functionalities. The experiments included live injection of artificial data into the free space detection algorithm run on the Nvidia Drive PX hardware. Velodyne VLP-16 [18] was the LiDAR sensor fully substituted by the prepared algorithms. Experiment included also the comparison of the free space detection functionality run on the real and synthetic data.

The c# and Python environment, which processes the raw LiDAR point cloud, were developed for the research work. That environment reshapes the raw LiDAR data into the format which is usable in the supervised learning. For the next experiments, which included training of the deep neural networks, developed scripts saved each point cloud frame and then reshaped it into the range image [21]. Those novel method not only simplifies the neural network architecture but also speed up the inference. To compare the performance of the neural network models trained on synthetically generated data, the real data from the Waymo open dataset [3] was used as the reference data. The Python environment processes the real Waymo data in the same way as the synthetic data. Developed tools build proper labels for all the objects of interest, which fit to the size of the generated range images.

In this work the new method for automatic labeling of objects from simulation was introduced. First experiments, where the objects were located in the point cloud based on the position in the simulated world, showed poor precision of the labeling. That's why the novel method, which uses the segmentation LiDAR sensor, available in the newest versions of the Carla simulator, was developed. The segmentation process merged with the ray-casting algorithm is able to produce the data, where object labels are accurate. Conducted experiments present the comparison of both developed methods of automatic labeling.

To evaluate the quality and usability of computer simulation and synthetic data generation, one of the best neural network detectors was used. Detector is called *You Only Look Once* (YOLO). Fourth version, YOLOv4 [22], of network was used to train the model and compare the results out of different custom datsets. Developed environment allowed to prepare a few custom datasets. That datasets included thousands of range images generated out of the Carla simulation scenarios and also thousands of Waymo real data reshaped exactly the same way as the synthetic data. Carla simulator allows to configure the LiDAR sensor model. Correct configuration of the sensor was crucial to achieve high performance. Data must fit into the characteristic of the top LiDAR on Waymo vehicle to minimize the synthetic to real gap. Fine tuning method was chosen to speed up the process of training. Pre-trained model weights *yolov4.conv.137* were used as the input for the training process. The whole training process, for all the models, was performed with the usage of the open access neural network framework called *Darknet*.

The last part of the work consisted of a few experiments, where different detectors were trained. The goal of the detectors was to predict the position of the pedestrian on the range image,

generated out of the LiDAR point cloud. All the detectors were validated on the same real data prepared out of the Waymo dataset. The obtained results made it possible to compare the performance and differences between models, which were trained with the synthetic training data only, real data only and with the use of the mix of both real and synthetic training datasets. To properly compare the trained models the evaluation metrics were calculated. Those metrics are based on the number of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) in a confusion matrix. The research included the analysis of the impact of the dataset sizes on the performance of the detectors for the real, synthetic and mixed data.

All the identified weak and missing points of the proposed and evaluated methods are described in the conclusions and result summary. Those are the main findings of the research work, which lead to the further improvements aimed at developing the technology of substituting the real data with the synthetic data in the automotive industry.

2.1 Structure of the work

The work is divided into the 8 chapters.

The first chapter is the "Introduction", which describes the background of the research. In this chapter the necessity to involve the complex computer simulations into the process of development of the new ADAS functions is explained. The second chapter concerns the scope and thesis of the work.

The third chapter contains an overview of already conducted researches in the field of autonomous driving and vehicle perception systems. In this chapter pros and cons of the methods and results published in the related works are discussed. Related works are divided into three major categories. The first one contains available road simulators, which are used for ADAS or perception systems validation and development. The second refers to the publications, in which the hardware-in-the-loop systems for ADAS were proposed. The third category includes the publications, where the simulation data was used to train the supervised learning models.

The fourth chapter contains description of the proposed and developed HIL setup, where ADAS camera sensor is put under test. The tests included TSR and LKA validation to indicate the performance of the sensor on the artificial images generated by Carla simulation. Results of the performed experiments indicate the value of synthetic data in terms of ADAS functionality.

In the fifth chapter the LiDAR sensor simulation was used to substitute the real LiDAR data. Described experiments involved real-time injection of the point cloud into 3D rendering software called VeloView. Further experiments proved usability of the synthetic LiDAR data in the real hardware applications, where the Velodyne VLP-16 sensor was replaced by the synthetic Carla data.

The sixth chapter concerns a study on synthetic LiDAR data used to trained deep neural network model. The goal of experiment described in this chapter was to determine the usability of the artificial LiDAR data in terms of real application. Detectors trained on the state of the art YOLOv4 model were compared to the models trained on the real data only.

The seventh chapter contains a summary and conclusions of the work. The last eight chapter is a bibliography, where all references are gathered.

The following main thesis was formulated:

High quality 3D rendering engines with complex and highly configurable sensor models are capable to generate synthetic data, which can support or improve the performance of the advanced driving assistant systems, if those relay on supervised learning technology. The input for those algorithms should be generated by camera or LiDAR sensor.

To support the main thesis two auxiliary thesis were formulated:

1. Synthetic data can be treated as a valid input for the machine learning model, if the model trained with those data indicates higher performance than the one trained on the real data only.

2. Simulation data can be treated as a valid input for the advanced driving assistant systems, if the results of the same task, in controlled conditions, are constant and repeatable.

3 Related works

ADAS validation is a time and money consuming process. That is why the automotive industry requires the solution to test their developed system, but without the necessity to drive around and look for the proper scenario at the moment. What's more, some of the rare but happening situations are impossible to be recorded or at least really risky to be arranged. A lot of tests is done on the closed test tracks by vehicles equipped with the newly developed functions. Test track driving has also it's disadvantages. A lot of work has to be put to create the realistic scenario, where traffic with other road participant is available. That's why the functionalities like AEB or LKA are mostly tested on tracks, since they do not require any road participant apart from the ego vehicle. Those problems can be eliminated by road simulation, so in the recent years, plenty of publications has been devoted to tackling this topic.

3.1 Newest solutions in road scenario simulation

Computer simulations are popular and helpful source of data. The idea to use the computers to simulate physical environment is as old as the first computers. The key feature in all kind of simulations is that it always simplify in some way the real process. Better simulator means that it simulates processes more precisely. In the automotive industry simulations were mostly used to design mechanical parts of the vehicles or to calculate the dynamics of the vehicle. For the last ten year the range of applications, where simulation are used in the automotive industry, has grown significantly. This changes origin form the moment when the available sensing technologies enabled to think about systems which help driver to drive more safely on the road than humans. Those driving assistant systems technologies gave the opportunity for the industry to think seriously about autonomous driving. Nowadays any new car, which has an European homologation, has to be equipped with the AEB and TSR systems. That is the point where the industry started to support their newly developed system with road simulators. That kind of simulations can significantly reduce the time and expenses required to develop the new features, but there is one big problem. As it was already said, simulation is just an approximation of the real world, and this started to be the key factor to be improve. Since all modern vehicles are equipped with cameras, the main goal of the simulation is to faithfully reproduce the road environment. Camera images are not the only one, which are used for the vehicle sensing systems. Ultrasonic, radars, LiDARs or Far-Infrared sensors also start to be the part of the perception systems. Most of the modern ADAS relay on the technology of supervised learning and, therefore, the differences between real and simulated data are crucial. Best quality and high performance with low gap between synthetic and real data is required from the simulators.

One of the first examples of the road graphics used in terms of ADAS in the automotive industry is described in Gietelink publication [6]. First attempts, where road environment was rendered in 3D, had a really poor quality. Those simulations were mostly useless for the purposes of camera or other sensor data generation. Computer game industry, totally unrelated to the automotive industry, gave the chance to solve the problem with road rendering quality. For the computer games the best rendering engines, with highest performance are developed. That's why, top state of the art simulators for ADAS validation are using the Unity or Unreal Engine 4 to deliver 3D, high quality data [23]. It is worth to mention that some of the projects like Carla are aiming to work on the newest Unreal Engine 5, which delivers almost photo realistic representation of the scenarios

and state of the art performance.

Nowadays, there are many simulators for virtual validation of the autonomous vehicles. Some of them are popular in the automotive industry like PreScan [24], CarMaker [25], rFpro [26] or CarSim [27]. But a lot of those simulators cores is having outdated rendering engines. The highest quality of rendering is provide by simulators supported by the Unity or UE. Carla [19], Nvidia Drive Constellation [28] or AirSim [29] are built on the UE4 and for example LGSVL [30] is built on Unity. All referred simulators have their own unique features, all provide road simulation but with different characteristics and maturity of the project. Figure 3 depicts four examples of images generated out of simulators.

Figure 3: Four examples of images generated by different simulators. 1 - CarMaker, 2 - PreScan, 3 - Nvidia Drive Constellation, 4 - Carla.

For example Nvidia Drive Constellation is focused on the cloud computing and scalable environment to meat ADAS safety levels in terms of validation. Nvidia solution is not an open source project, it is rather created for automotive OEMs to help them with development of the new features in vehicles. Microsoft AirSim is an example of the open source project, which turned to be the main issue. Low interest and small community around the project made the AirSim unusable software with no support and further improvements. The best choice for the purpose of this work was Carla. Carla is an open source simulator developed to support training, prototyping and validation of the autonomous driving systems. UE4 rendering engine delivers highest available quality of the images and, together with the ray-casting ability, it is able to simulate range of suitable sensors. Carla project focuses on the realistic modelling of the sensors used in vehicles, which is the key factor why this simulator was the best choice for this research. Moreover, the project is constantly under development and provides massive improvement in the newly released versions. The huge community on git-hub is also a big plus for the project, especially if any modifications of the original program are needed. As the Carla is based on the UE4, it is easy to prepare new scenarios with the usage of unreal engine editor. Simulator in version 0.9.12 already includes 7

diverse maps, where the vehicle can drive and record data out of the sensor. Carla comes with full Python interface to control and manipulate the simulation. In this research, the needed part of this interface was retyped in c# language. c# language was the best possible connection between Carla and the automation system used by HIL, which is introduced in the chapter 4. The best features of Carla simulator enabled the researcher and this work to experiment with highly realistic data generated out of camera, LiDAR or radar sensors.

3.2 Hardware-in-the-loop setups

First of all, there are several publications, which present the HIL or VEHIL approach to validate the ADAS hardware. In [32] publication the authors proposed VEHIL setup, where ADAS aftermarket camera Mobileye 560 was attached to the front windscreen of the vehicle. The whole vehicle was situated on the chassis-dynamometer and in front of the Camera view the monitor screen was attached. The proposed setup made it possible to simulate vehicle dynamics in the Pro-Sivic simulator with some simple traffic scenario. The only ADAS aspect of those setup was that Mobileye camera, thanks to which it was possible to detect pedestrians or vehicles out of the simulation image. Detection information were only indicated on the internal camera display. Similar setup, where Mobileye 660 camera detects speed limit signs, is developed and validated in this work. But the main difference it that the HIL setup described in the chapter 4 does not need the vehicle itself. What's more, the whole detection process is statistically proven to be deterministic and repeatable. That type of experimental analysis is not provided in the Rossi et al. publication.

In the publication that was introduced in 2016, Sandalek et al. [15] successfully performed the ACC test. BMW engine on the test bench and simulated radar sensor in CarMaker were connected together to perform the closed loop validation of the engine fuel consumption within known simulated road track. In case of this publication more focus was put on the validation of the combustion engine. Simulated radar sensor did not require and graphical and high quality data to output the distance to the proceeding vehicle. The chapter 4.5 of this research concerns the HIL setup, where the closed loop test on the ADAS LKA was successfully performed. What's more, that experiment was the first described CL-HIL test, where simulation data was generated in the real-time and the system reacted on the ADAS camera detections to control the vehicle steering. This experiment required high quality rendering abilities, to let the sensor properly detect objects.

The novel solution VEHIL of steerable rolling test-bench is presented in Solmaz and Holzinger publication [16]. Similar to the Rossi et al. publication, the whole vehicle is put on the test-bench to validate the ADAS functionalities. Solmaz and Holzinger test vehicle is a properly equipped with the hardware. The vehicle includes 7 cameras, 6 radars and LiDAR. Additionally, in the trunk, there is the whole setup of computers and measurement devices. Proposed VEHIL was used to demonstrate working ACC and LKA on the test-bench. Setup for the ACC is almost the same as the one is Rossi et al. publication. Simulation manages the vehicle speed and accordingly to the simulated radar data the distance to the proceeding simulated vehicle the test-bench is corrected. It's worth to mention that the radar sensor is not modelled in the simulation at all. Only the communication outcome of the sensor is simulated, which means that the setup cannot simulate any road environment disturbance which might occur on the real road (e.g. the interference between other objects in the scenario). The second experiment is more interesting: the rolling test bench reacts to the detections of the Mobileye camera and steers the real vehicle wheels. The test bench proposed in the research has internal structural delays, which result in the oscillations of the vehicle steering.

These kind of problems on the HIL system might cause incorrect results of the ADAS function validation. Therefore the proposed method with vehicle is not fully useful in terms of reparable and consistent ADAS testing. Experiment results described in the chapter 4.4.1 of this thesis didn't show any structural delays of the test bench. Test automation and measurement environment is controlled by the real-time core, so no mechanical part can influence the results of the test.

In 2021 the Kaseem et al. published the paper [33] where camera HIL is introduced. The setup includes simple USB Camera, projector screen and a steering wheel to control the simulation run on the computer. Researchers used unreal engine to produce the simulation image for camera. The algorithm implemented in the Matlab environment gave the setup information about line markers visible on the simulated road. Publication includes only one experiment where the camera sensor detects line on the road and, on this basis, the algorithm is checking whether the ego vehicle is not crossing unintentional the lane. This type of ADAS functions is known as a lane departure warning. This simple HIL example might be compared to the one developed in this work, but there is no evaluation of the real ADAS sensors and setup is not capable to the work in the closed loop. In the further chapters of this work much more complex and realistic variants of ADAS validation are experimentally verified.

3.3 Synthetic datasets in supervised learning technology

Synthetic data of road simulation is not only useful in the field of validation or testing. The second important task is to support or even improve the machine learning model performance. Attempts to substitute the real dataset with the synthetic one in case of images or other type of data lead the researchers to the phenomenon known as *synthetic to real gap*. This problem is well explained in Nvidia and Fabbri et al. [34, 9] publications. The cost of creating the new datasets out of the real sensor records is huge, especially if it has to be the totally new data set, which happens in case of the sensor characteristic change. One of the ways to solve this issue is to apply the fine-tuning method. It allows to train deep neural network with the new data without forgetting everything what was learned from the old training data. This approach is valid only if the task for algorithm is the same or at least similar to the original one.

Probably the biggest available and the best described synthetic dataset is called MOTsynth [9]. This dataset includes almost 1.4 million of frames synthetically generated. To produce this high quality dataset researchers used GTA V, one of the most popular computer games. The reason to use such a game is that it provides large realistic world with full traffic simulation. The world includes divers city scenarios with hundreds of actors like vehicles, pedestrians, bikes and other. With the usage of this dataset researchers performed variety of transfer learning experiments. The result showed that it is possible to bridge the gap between the real and synthetic data. It's worth to mention that the authors of MOSTSynth dataset focused on the detection of pedestrians out of the camera images. In this work there are no attempts to reproduce the results of MOTSynth authors in case of camera sensor images. The research is more focused on experimenting with synthetic LiDAR sensor data. There was no sense to tackle again the same camera sensor topic.

There are more publications where the GTA V computer game is used to generate the synthetic data. In Lanzi et al. [11] publication this game is treated as the source of the pedestrian models. This research is focused on the problem of detecting the people body position based on the body joints. Similarly to the previously mentioned publication [9], researchers successfully

extended the real dataset with the synthetic data and improved the original real MOT-16 dataset performance. Again the input data was the colour image, where object were to be detected. It's worth to talk about the usage of GTA V game in such researches. Of course the game was not developed for the purposes of machine learning technology improvements. To let it work in such a way, the Scrip Hook V pack has to be used with the game. The point is that the developer of the game *Rockstart North* company, banned this pack in the past, due to the copyright infringement. Probably the usability of this simulator for further experiments might be inhibited.

There is one more important publication of Nvidia company [34], where the new idea, how the synthetic data can help to improve or beat the real data, is proposed. The researchers built their own simulator where objects of interest, for example cars, are randomly placed together with other 3D objects on the totally imaginary backgrounds. This approach is an opposition to the experiments where the photo-realistic images are generated. Thanks to the introduced method, the synthetic data forced neural networks to detect only the shapes of the objects of interest. Researchers from Nvidia proved that this method can beat the networks which are trained on the real data only. It is clear that the abilities of the modern 3D rendering engines are good enough to support the supervised training technology. The open question is if these success can be also replicated for other types of sensors, like LiDARs.

There are a few publications where the synthetic point cloud generated out of the LiDAR sensor is used to train sensing algorithms. The first example of such an attempt is Wang et al. [35] publication. In this publication Carla simulator is used to generate the synthetic point cloud data. The synthetic LiDAR point cloud is used to train a deep neural network, which is compared to the models trained with the data of real KITTI [4] dataset only. The researchers achieved the highest AP average precision for the mix of synthetic and the real-dataset. The experiment involved only 1,7 k of point cloud frames, so the results were not the best. It is important that the input for those experiments was 3D, due to the usage of a raw point cloud data. This additional dimension makes the task more complex and it's harder to train the neural network properly in comparison to the input data in the form of images which contain 2D information only. One more relevant information is needed to understand the results described in this paper. Older than 0.9.10 versions of Carla have simplified collision boxes for the actors. Since the Wang et al. work was published in 2019, they had to use the version with that type of collision boxes. Better quality of the collision boxes in the newer versions of Carla significantly increases the variability of the LiDAR simulations. Simple collision boxes mean that the synthetic data does not look like the real one, and this makes4 a big difference. In the chapter 5.2.1 of this work this difference is precisely described and shown on the examples. The experiments evaluated in this work include results obtained from one of the newest versions of Carla simulator.

Already mentioned ability of GTA V game to simulate realistic traffic scenarios was again used by the B. Hurl, K Czarnecki and S Waslander. In Hurl et al. publication [36] they used ray-casting method which is native functionality of this game to produce point cloud out of the game streets. They found that native ray-casting has a lot of limitation and is not useful to create similar point cloud like for example Velodyne lidar HDL-64E from KITTI dataset. They created own algorithm of ray-casting to produce comparable with real data. As the result they experiment to train pedestrian detector. Similar to the Wang et al. research they chose to detect object out of 3D point cloud. The outcome, bounding boxed, of the algorithm is also 3D. Additional dimension makes task harder so the result are not high. Almost 60% of the AP for the best model trained with

the mix of synthetic and real data and few percentage less achieved on real dataset. In chapter 6 of this work, provided lidar model is much more realistic than the one in Hurl et al. research. Carla allows to configure lidar sensor with dozen of parameters to fit the output data into the real sensor data. Proposed in Hurl et al. ray-casting algorithm generates perfect point cloud, which real sensor wound never generate. The reason is that reflective of objects might differ which causes the drops of points on some specific surfaces. Researchers also explains the similar issue with the object and sensor data synchronization. In case of fast rotation of sensor the synchronization between data and scanned scenario must be perfect. If not, scanned object might be misshaped or the automatic labeling of those objects might be imprecise. The same issue occur in experiments conducted for this work, but the new method with semantic LiDAR segmentation was introduced to overcome those problems. New method to solve the problem is described in chapter 6.6.3.

In 2021 Spiegel et al. published the paper [37] where deep learning algorithm was trained on the 3D point cloud. Same as in Wang et al. research, Spiegel et al. used Carla simulator to generate the point cloud. Their task was to perform the segmentation of the objects scanned by the LiDAR. Each sensor scan has the noise on range axis added and then saved into one huge point cloud to build up the whole map of the Carla simulator. Accumulated data were used by researchers to train KP-FCNN architecture of the neural network, built for the segmentation task. The results showed 93.8% of overall accuracy, which indicated high performance of the algorithm. The highest results were achieved for the objects like building, ground and vegetation. Those object are the biggest ones and are the main part of the scenarios. For example miscellaneous objects has only 46% of accuracy. Publication of Spiegel et al. did not try to estimate the gap between the real and synthetic data. Experimentally trained model was evaluated only on synthetic data, so there is no information how such a model could work with the real LiDAR point cloud in terms of object segmentation. The experiments shown in chapter 6 of this thesis compares performance of trained models trained synthetic and real data. This method provides direct information, whether the synthetic data is usable in real applications.

In 2019 Dworak et al. published the paper [38] where the idea to use synthetic data of LiDAR sensor is experimentally verified. Researchers used Carla simulator to generate 3D point cloud and feed deep neural network. The task for the algorithm was to detect pedestrians out of the road scene. The input, similarly as in all previous publications, is also 3D and the output is 3D bounding box. The experiment was evaluated on real and synthetic data from KITTI [4] dataset. The results described by Dworak et al. are a perfect example of the real phenomenon called *synthetic to real gap*. Models which were trained on Carla data and evaluated on KITTI data has much lower AP than the one trained with the usage of KITTI. The differences are huge: the model trained with the Carla data only achieved 25% of AP, while the same model trained on the real KITTI dataset obtained 79% of AP. The root cause of this results is described by the researchers in [38]. As already mentioned Carla versions lower than 0.9.10 have collision boxed simplified to just straight cubic shape. This issue is precisely described in chapter 5.2.1 of this work, where the examples are depicted. The solution for this problem is crucial in terms of bridging the gap between simulated LiDAR data and the real one.

Experiments described in the referred papers were the core source of the information about already known methods and problems in the field of feasibility study on the synthetic data. Similar problems and challenges appeared within the course of the research presented in this thesis. The focus was put on understanding and overcoming these problems, developing the solutions if pos-

sible, and proving that they are applicable in the real road life. The goal is to improve the ADAS function with the usage of simulated data, which might vastly decrease the expenses and the work load on the newly developed algorithms. The main goal is to increase the public road safety and decrease the impact of imperfect human drivers onto that safety.

4 Vision sensor validation on HIL

4.1 HIL test object

The first experiments in the course of the carried out research were aimed at testing the variability of simulation graphic abilities in terms of ADAS sensor functions. Research works stared from the task consisting in selection of the proper and adequate test object, on which the experiments might be conducted. Since the core sensor used by industry for ADAS functions is a camera sensor, as the first test object an ADAS camera was selected. Due to the IP rights and legal rules there was no option to take one of the cameras from the production car. That's why the best and almost the only choice was the aftermarket camera provided by Mobileye company. Those cameras are widely used by the researchers and provide almost the best available performance and functions available in the nowadays cars. In the chapter 4 there are provided publication examples [32, 16] where Mobileye cameras were used. The same aftermarket camera Mobileye in version 660 was selected for the HIL as the test object. Figure 4 depicts the Mobileye 660 Camera.

Figure 4: Image of Mobileye 660 ADAS camera system.

Additionally, this sensor has an open CAN communication, which means that all the detections done by sensor are provided over CAN messages. This sensor includes three main ADAS functionalities: AEB, LDW and TSR. Mobileye developed this system in order to deliver the ADAS camera based functions to the cars, which are not equipped with such functions originally.

4.2 HIL automation platform

The second most important part of the HIL is the software environment, where all hardware and software devices can be connected and managed. There were a few requirements to be met by this system. First of all, it has to work on standard computer machine and Windows operating system. The HIL solution should be easy and handy to use for the testers. Moreover, there was one more, but very important requirement: the whole system should support the real-time measurement abilities, and preferably, it should achieve it without any external hardware. Investigation into existing automation systems that can work as an HIL environment platform resulted in the list of available software platforms for such projects. The list includes popular software like LabView with Python integration toolkit, Matlab Simulink with ROS toolbox, or AutomationDesk developed by dSpace company, but also less popular solutions like ViCanDo or Robotic operating system run on Windows subsystem for Linux (WSL). Most of the best platforms are commercial products, which require the yearly payment of the licenses. Each of those solutions has some advantages and disadvantages, but there was one additional solution, which fitted the research requirements

best. Morphee [20] was found by French company and later bought by FEV company. For over 20 years it automates and controls the engine test benches. It's native ability to support the real-time measurements within real-time operating subsystem called RTX [39] is crucial for this research. What's more, this RTX subsystem does not require any additional hardware to be run. Those unique features and the fact that FEV Poland is one of the IP right shareholders of this work, makes Morphee the best choice. Figure 5 shows an example application of the user interface created in Morphee for this research.

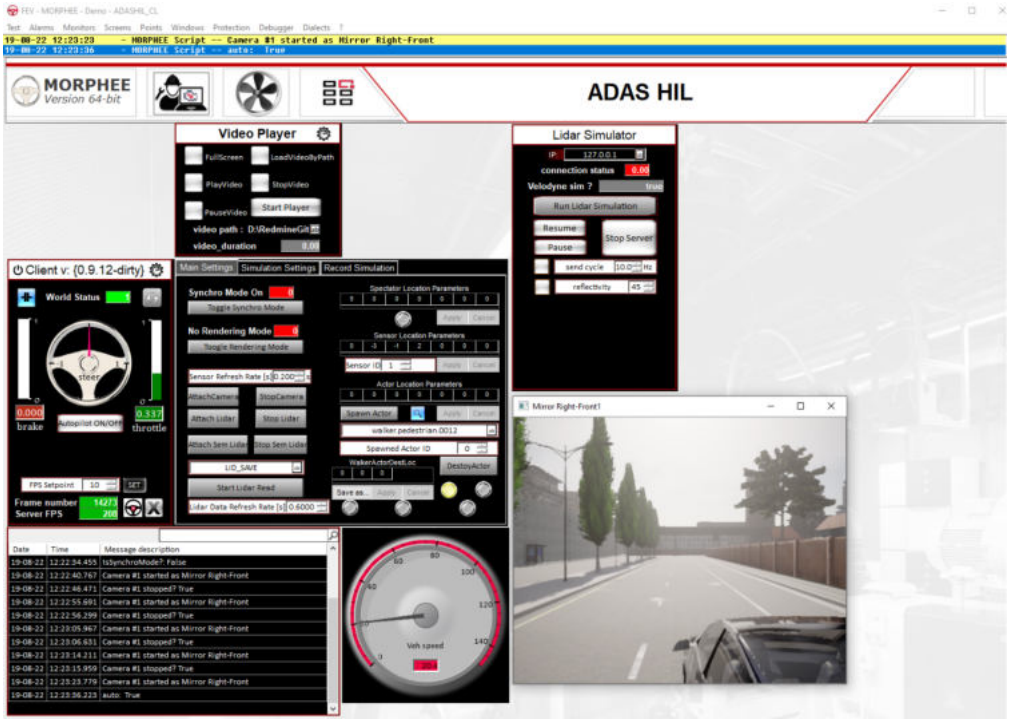

Figure 5: Morphee example of user interface created for one of the experiment.

4.2.1 Description of Morphee RTX subsystem

RTX subsystem is an important part of the Morphee platform and, therefore, it's worth to understand how it works. As everyone knows, the Windows operating system has no real-time capabilities. That's why Interval Zero company has developed the RTX subsystem, which enables the real-time performance of the application run on the Windows side. The processes of Morphee are taken out from Windows side to the RTX kernel, which is the real-time operating system. That's how Morphee, which is an application available for Windows operational system, has the real-time abilities. RTX has a lot of dependence on hardware parts of the computers and that's why not every PC is able to run RTX. The first and the most important issue is that PC has to be equipped with the multi-core processor. The point is that, in order to separate the non-real-time and real-time processes, RTX needs to conquer totally the part of the computer resources. It means that the part of the processor cores must be taken by RTX. The same is done with computer RAM. Part of it has to be fully managed by the RTX subsystem. Figure 6 presents a diagram which explains how the RTX subsystem cooperates with PC hardware parts and Windows system.

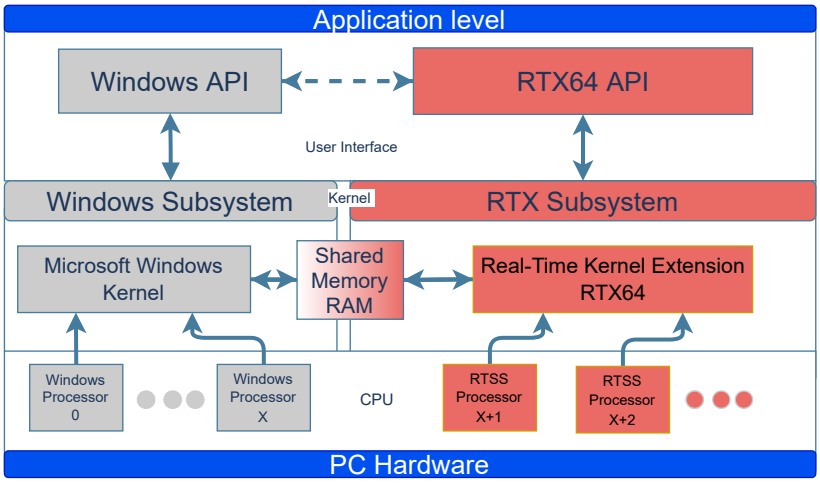

Figure 6: Diagram explains how RTX and Windows cooperates within the same computer hardware.

On the diagram it's depicted that some cores (e.g. one core of the processor) are taken by the RTX64 kernel. With this core resources the processes are separated from the Windows system application layer. This allows the RTX to run and manage e.g. the real-time measurements done by Morphee. Part of the PC memory is shared and, therefore, the Windows and RTX kernels are able to communicate and transferred data between them. What is the advantage of the fact that Morphee is run on the RTX? The whole Morphee configuration is run in so called tick mode. This one tick is a constant, configurable time when all the measurements, variables and processes are updated within setup. Proper Morphee setup can work in the 2 kHz tick, which means that every single operation will be done within half of the millisecond and, of course, hundreds of calculations can be performed in each tick. Thanks to the real-time ability of Morphee platform it is possible to build reliable and repeatable results of the ADAS tests.

4.3 Architecture of the HIL system

The last step to run the vision based experiments on Mobileye camera was to connect the parts of the HIL system. Camera requires the input image and the CAN communication with the vehicle. Both parts have to be delivered by the Morphee core. In the course of the experiments, the image input for the sensor was delivered by the computer display. Display attached to the frame together with the camera makes it possible to fix the position of the sensor in front of the screen. Such frame and display setup was enough to calibrate the sensor. For the purposes of object detection, the image visible to the sensor has to be precisely positioned. This requirement is necessary to receive correct detection out of the ADAS functions of the sensor. Calibration of the sensor was done with the usage of manufacturer software called ME toolkit. Software receives the image stream from the sensor, and this image is treated as a source of the information about camera position. In standard situation it is position of the sensor in the car, but in this HIL setup it is the position in front of the display. Figure 7 depicts the camera sensor attached to the frame and aimed into the computer display. Source of the image [40].

Figure 7: Left: HIL frame with Mobileye sensor and display attached. Right: Image stream out of the sensor from the same moment of time as on the left image.

In 7, on the right image, there is a red cross in the middle, which should aim to the middle of the road. Of course the best scenario for sensor calibration is straight road like the one visible on the left image. It's also important to set up the level of the red cross at the same position as the horizon of the road. Sensor calibrated in such a way gives the best performance in terms of object detection and distance measurement.

Establishing CAN communication is the last step required for the proper work of the sensor. Usually the aftermarket cameras are connected with the car where they are mounted. In case of the considered experiment, it is the HIL setup that must simulate the vehicle - sensor communication. For these purpose the popular software which is compatible with Morphee was used. It's is called CANoe, and it's a product of Vector Gmbh company. Application of CANoe together with the dedicated hardware *Vector box* made it possible to simulate communication with external devices, for example via CAN protocol. CANoe with hardware are able to generate CAN messages which are normally sent by the vehicle and deliver it to the sensor. Thanks to CANoe - sensor communication it is also possible to receive CAN messages send by the camera. Such setup make the Mobileye to "think", that it is connected to the real vehicle. To let the sensor work properly it is necessary to provide information about the vehicle state. Most important is the vehicle speed. On this basis, AEB is calculating time to collision. Information about the actual state of indicators is also needed to let LDW function work as expected. There should be no warning that vehicle is crossing the lines on the road if the indicator for that direction is switched on.

There is one more part of the developed setup, which is important for the data flow. The messages received and send by the sensor, should be transported to the Morphee software, since it is the core part, which manages the process of experiments. For the purposes of massages transportation the Fast Data Exchange (FDX) protocol [41] was designed. FDX allows to transport for example the data out of CAN messages with 1 kHz cycle time. That's how Morphee delivers and receives data from the sensor. Developed setup manages data flow between components with real-time performance. Figure 8 shows diagram of connections between HIL setup components.

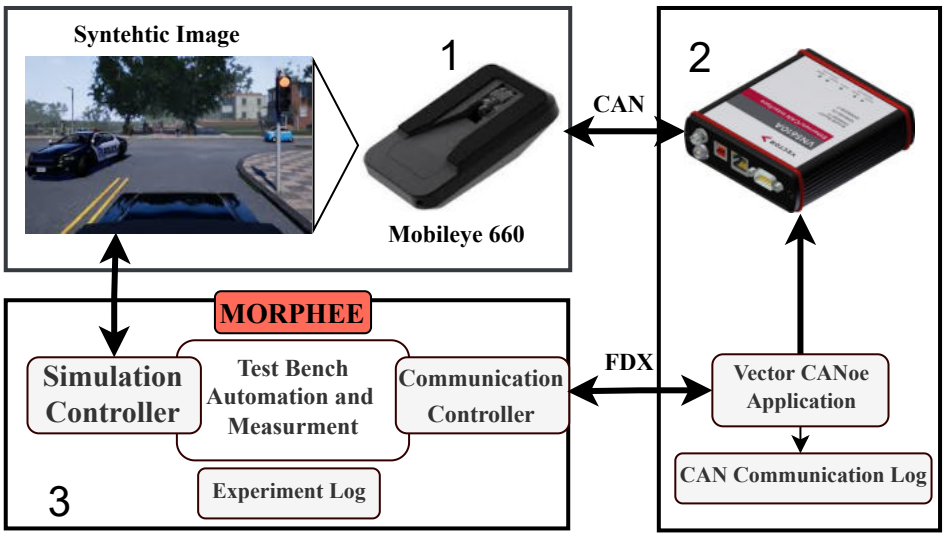

Figure 8: Connection diagram of the three most important elements of the HIL setup.1- Hardware part of setup, sensor and screen attached to the frame, 2- Can communication block, Vector Box VN5610A plus and Canoe application, 3- Morphee automation platform.

4.4 Real-time object detection experiment, run on simulated road scenario

The first experiment, run on the developed HIL system, was the test of TSR functionality delivered by Mobileye camera. The goal of this experiment was to determine if the synthetic road scenario can be used as an input for the ADAS sensor and whether it delivers stable, consistent and reliable results. With good enough quality of the input data, the sensor should correctly detect road signs even if it was trained on the real data only. Not only input data is relevant to get stable and repeatable results. The performance of the HIL setup is also important factor. Researchers in [16] has described that internal HIL setup delays caused the oscillations, which affected the test results. That's why it's important to defined the performance limits of the setup to be able to correctly distinguish between test object and HIL setup failures. One of the easiest task for ADAS camera is to detect speed limit sign. Internal inconsistency of the sensor functionality might add unwanted fluctuation to the test results, which, in turn, might affect the overall evaluation of the HIL setup. That's why the speed limit detection scenario was selected for this experiment. The experiment is an example of the open loop (OL) validation, where data output by the test object has no influence on the further test scenario. Since the OL-HIL test is considered, it is meaningless to lively render the input images, especially if it is planned to show exactly the same scenario many times. Therefore, the video stream recorded out of Carla simulator was used as the input for this experiment. In 9 the flowchart of the experimental tests is presented.

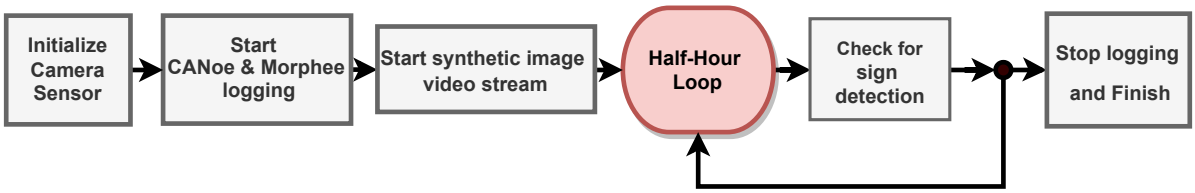

Figure 9: Flow chart of the ADAS TSR functionality test.

The first stage of the experiment consisted in initialization of the setup and camera sensor. After the initialization, measurements and logging started in CANoe and Morphee application. The main part of the experiment was an 30 min loop, where the same looped scenario was shown on the HIL display. Recorded video was 9 seconds long and included two different speed limit signs, 30 and 60 km/h. With such preconditions the expectation was that 30 min experiment should log around 400 of speed limit sign detection. Example screen of recorded video is depicted in figure 10.

Figure 10: Sample image out of the video prepared for TSR test experiment

sThe goal was not only to check if the setup is able to detect signs out of the synthetic image, but also to calculate the precision of the setup measurements. For that purpose the external, precise clock was used to calculate the time delays between the exact time of sign detection send over CAN message by sensor and the time when it was logged by Morphee. Since Vector box hardware used for CAN communication included FPGA clock on board, this time was used as the reference for the Morphee clock (Morphee clock is taken from one of the CPU cores). Thanks to this reference clock it was possible to calculate the exact delays, which may occur internally within the HIL setup. It is important to highlight that Morphee tick for this experiment was set to 1 kHz, so the measurement resolution is 1 ms.

4.4.1 TSR experiment results

As the output of the performed experiment two log files were obtained, which included information about the ADAS sensor detection of signs and the time stamps indicating when this event happened. One log was done with the FPGA clock time stamp and the second one by Morphee automation platform. According to the log files, sensor correctly detected 408 speed limit signs. Figure 11 depicts chart of the detection logged withing the experiment. For better visualization, each detection is a value of time which elapsed from the last detection.

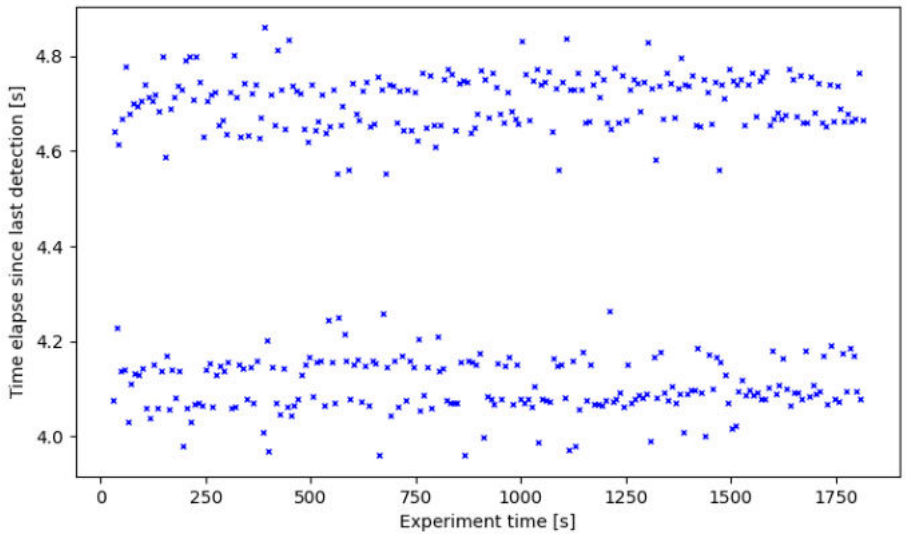

Figure 11: Experiment result chart, where function of time elapsed since last detection to experiment time is depicted.

In figure 11 it is clearly visible that the input scenario included two speed limit signs. First one was passed by the vehicle after around 4 seconds and the next one at the end of the video. The fluctuations of the detection time appeared, since the sensor each time calculated the point where the vehicle crossed the sign to indicated the actual speed limit of the road. That is how the TSR functionality works in the Mobileye ADAS camera. Results show that synthetic scenario prepared for experiment is good enough to let the sensor detect all visible signs. The second goal was to determine the precision of the build HIL setup in terms of timing and data flow. To define the internal HIL delays, the difference between the real time of sent CAN messages and the time of detection logged by Morphee was calculated. Figure 12 shows the time differences that occurred between reference FPGA clock time stamps and the Morphee clock time stamps.

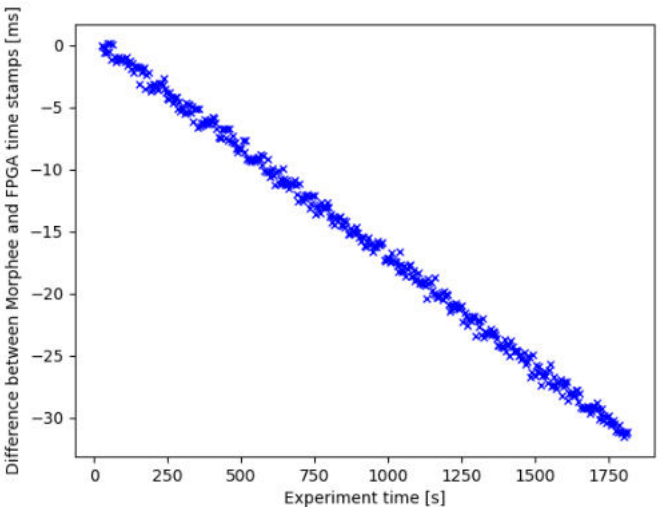

Figure 12: Difference between reference clock time stamps and Morphee clock time stamps for all detected speed limit signs.

The chart presented in figure 12 shows unexpected result, where the difference between FPGA clock and Morphee clock is increasing constantly for the duration of the experiment. The source of this surprising results is the clock drift. The drift phenomenon is the situation when one clock does not run at the exactly same rate as the reference clock. The difference between clocks is called drift. Out of the data gathered in this experiment it is possible to define what is the value of the drift between reference and the Morphee clock, and compensate the result by this drift. The results after the drift compensation are depicted in figure 13.

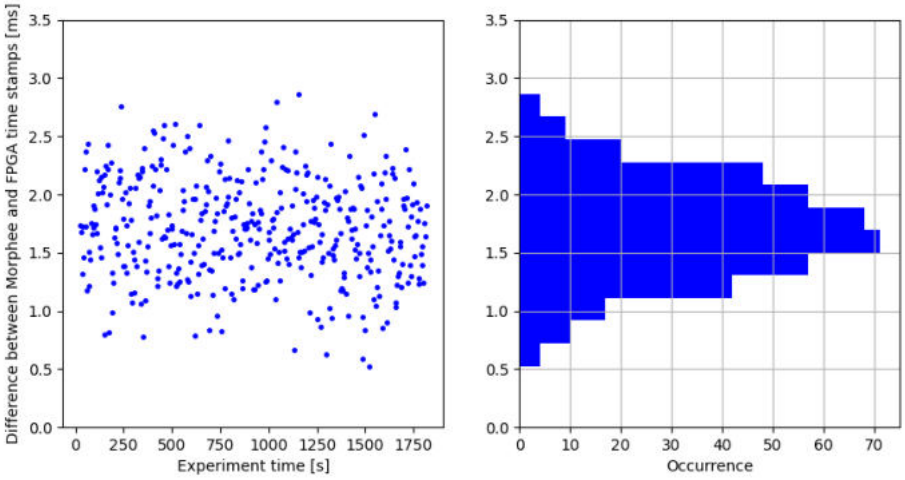

Figure 13: Left: Difference between reference clock sand Morphee clock after drift compensation. Right: histogram of the data depicted on the left chart.

In figure 13 it is visible that the maximal delay withing HIL measurement system is less than 3 ms, and mostly it's between 1.5 to 2 ms. Those low delays are achieved by Morphee platform mostly due to the fact that it is run on the real-time RTX subsystem. This solution where non-real-time and real-time application can work together on the same machine is unique. All competitive

HIL or VEHILE systems described in the publications [12, 15, 16] require dedicated hardware to enable real-time abilities or don't have any real-time solutions. TSR functionality experiment run on ADAS Mobileye camera showed that synthetically generated data out of Carla simulator can be used as a valid source of road scenarios and also that the results are stable, constant and repeatable. Further experiments, with more complex setup, were performed on the built setup to improve the evidence that high quality synthetic data can be used for ADAS validation and development.

4.5 Closed-loop Lane Keeping algorithm run on synthetic camera data

The next performed experiment where synthetic data was used included more complex closed-loop setup. This experiment was the first successful closed-loop lane keeping algorithm (C-L LKA)run on the HIL setup where the real-time, synthetic road scenario was used as an input image for the ADAS camera sensor [40]. The main goal of the experiment was not only to validate whether the Mobileye sensor detects correctly the line markings on the simulated road but also to use that detection to actually steer the vehicle. As the result of the research the safe and reliable environment was obtained, where newly developed algorithm can be lively put onto the road and validated. This type of approach can not only significantly reduce the costs of development of the new algorithms, but also reduce the work load in such projects.

4.5.1 Lane Keeping Algorithm

Mobileye 660 as ADAS sensor delivers functionality called Lane Departure Warning. Sensor algorithms detect the line markings to predict the line shape and, on the basis of this information, to calculate whether the vehicle is not driving out of its lane. If departure of the lane happens there is a warring sound which informs driver that the vehicle is crossing the lane. Mobileye sensor does not deliver pure LKA functionality but only detects the lane markings. That is why to conduct the experiment where the camera sensor lively steers the vehicle this algorithm has to be developed. The input for algorithm is the line detection and the output is the steering wheel position, which makes the vehicle to follow the road. The state of the art algorithms detect and describe the line markings as a third-degree polynomial model. This model is a function *Y(x)*, where *x* is the longitudinal distance and the *Y* is the lateral distance to the sensor from the line. The equation 1 is the definition of the third-degree polynomial line model.

$$Y(x) = C_r x^3 + Cx^2 + tan(Hx) + C_0, \tag{1}$$

where:

C_r = curvature rate [1/m^2]
C = curvature [1/m]
H = heading angle [radian]
C_0 = Distance to the line at x = 0 [m]

Mobileye 660 model provides a third-degree polynomial model of the lines. In comparison to the simplified second-degree polynomials, which were delivered in the older versions of Mobileye camera [12], new algorithm calculates the curvature rate for the detected lines. The difference makes the sensor to detect the s-curves, not only the parabolic shapes. The equation 2 defines the simplified model provided by the older versions of the Mobileye sensor.

$$Y(x) = 2Cx^2 + tan(Hx) + C_0, \tag{2}$$

Parameters of the line model are provided by the sensor over the CAN communication. Sensor calculates the parameters out of the image and then sends it every 60 ms. Camera sends also the information about the view range of the detected lines and the type of the line. The reference point for the parameters is set at the rear axle of the vehicle. The model of the detected line is the first step in the algorithm to calculate the steering angle. Sensor delivers the parameters of the left and right line model. The next step is to define the destination point. This is the point where the vehicle should aim to stay on the lane. Formulas 3 define how the destination point is calculated from the parameters provided for both detected line markings.

$$\begin{aligned} L_c &= \frac{(|C_{0l}| + |C_{0r}|)}{2}, \\ C_{0\Delta} &= C_{0l} - C_{0r}, \\ Y_l(x) &= C_r x^3 + Cx^2 + tan(Hx) + C_{0l} - L_c, \\ Y_r(x) &= C_r x^3 + Cx^2 + tan(Hx) + C_{0r} + L_c, \\ X_d &= \frac{Y_l(x) + Y_r(x)}{2} + C_{0\Delta}, \end{aligned} \tag{3}$$

where:

L_c = Lane center at x = 0 [m]
$C_{0\Delta}$ = offset from the center of the lane at x = 0 [m]
Y_r, Y_l = lateral distance to left and right destination point [m]
X_d = destination point [m]

Destination point is located in front of the vehicle, in the middle of the detected lines. Additionally, the offset position of the vehicle on the lane is added to the destination point to fix centering of the vehicle on the lane. The value of the x (longitudinal distance for the algorithm) is based on the view range defined by the sensor. If the view range is higher it means better visibility of the lane marking, so the destination point is calculated further in front of the vehicle. Destination point is used to define the radius of the curve for the vehicle to follow. Radius is calculated out of the circle formula 4.

$$Ax^2 + Ay^2 + Bx + Cy + D = 0, \tag{4}$$

Three points set on the circle are needed to calculate the radius of the circle. One of the points is the destination point, second one is the zero point (0,0) and the last one is selected as a reflection point to destination point relative to the lateral axis, see figure 14. Formula 5 defines how the radius is calculated and the parameters A, B and C are defined by formulas 6

$$r = \sqrt{\frac{B^2 + C^2}{4A^2}}, \tag{5}$$

$$\begin{aligned} A &= x_2 y_3 - x_3 y_2, \\ B &= (-y_3)(x_2^2 + y_2^2) + y_2(x_3^2 + y_3^2), \\ C &= (-x_2)(x_3^2 + y_3^2) + x_3(x_2^2 + y_2^2), \end{aligned} \tag{6}$$

where:

r = radius of the circle [m]

x_1, y_1 = point (0,0)

x_2, y_2, x_3, y_3 = coordinates of destination and reflection point located behind the vehicle, respectively.

Since one of the points is the zero point (x_1,y_1) = (0,0) the formulas for circle parameters are simplified. The example drawing of the detected line model and the estimation of the center line based on the destination point is depicted in figure 14.

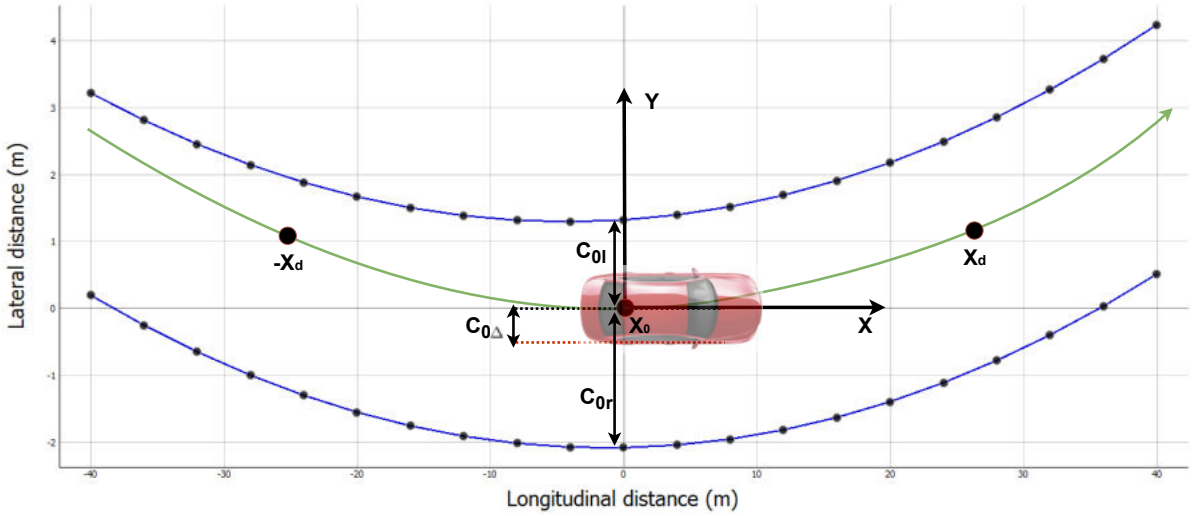

Figure 14: Example draw of the line marking models and the destination points location. Left and right lane markings are draw with blue color. Green line in the middle is the circle based on the three point X_0,$-X_d$,X_d. $C_{0\Delta}$ defined offset between middle of the lane and the middle of the real vehicle axle. C_{0l} and C_{0r} are the distances to the left and right line, at the position of the real vehicle axle.

On the basis of the radius of the circle visible in figure 14 (green curve) the steering wheel turning angle α can be estimated. The estimation is based on the distance between vehicle axle and the radius of the circle to be followed. Equation 7 defines how the steering wheel angle is calculated based on the circle radius.

$$\alpha = tan^{-1}(\frac{l}{r}), \tag{7}$$

where:

α = steering wheel angle [radian]

l = distance between vehicle axles [m]

Since the input for the simulator for steering is a range of (-1,1) values, the steering wheel angle has to be mapped onto this values. Function was defined as linear and the slope of this linear function was measured empirically in the simulation. The measurement based on the steering value and the angle of the steering wheels was performed in the Carla environment. Vehicle speed has

no influence on the developed LKA algorithm, but it is the information required by the sensor to properly detect lines. That's why vehicle speed is cyclically sent to the sensor.

Summing up, the Mobileye detects the third-degree polynomial parameters of both right and left line markings. Next step is to calculate out of the parameters the destination point and define the radius of the circle, which vehicle should follow. Base on the radius the steering wheel angle is calculated and, at the end, the correct value of steering is set to the simulation to control the vehicle position. Each 60 ms new steering wheel position is calculated and sent again to the simulation.

In order to explain how the developed HIL is setup to manage the data flow for LKA closed loop experiment, in figure 15 the diagram of the HIL components is provided.

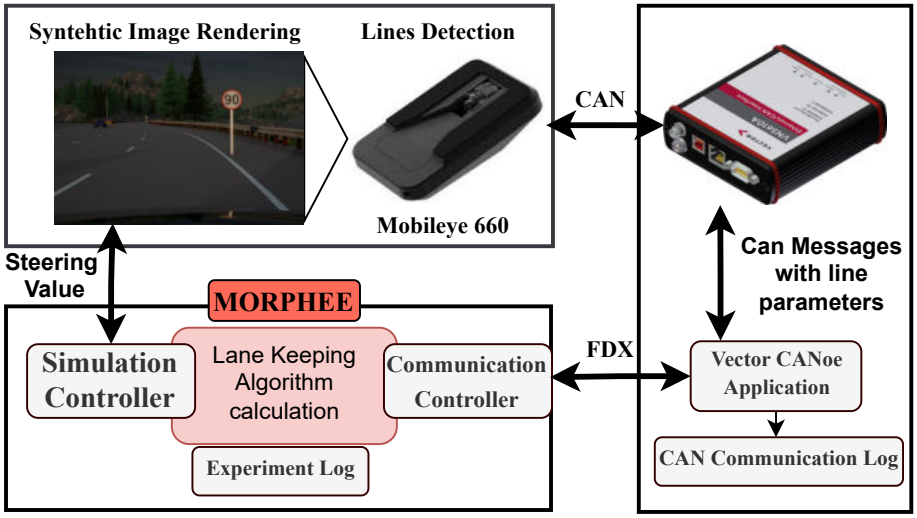

Figure 15: Diagram of the connections between HIL components setup for LKA closed loop experiment

On the diagram depicted in figure 15 there are (similarly to the previous experiment) three main blocks. The HIL frame sensor detects the line out of the image displayed on the computer monitor. Detection flows to the Morphee via CAN and FDX protocols. Morphee LKA component calculates steering value and with other vehicle controls (throttle, brake) sends it to the simulation. Simulation renders the new frames with road where vehicle moves accordingly to the control send from Morphee. This changed image is again an input for the sensor to detect line markings. The process repeats over and over to perform live validation of the LKA based on the synthetically generated data.

4.5.2 LKA experiment results

For the experiment one of existing in Carla map was used. This particular track was choose for this experiment, because it is a huge loop with different left and right corners. Track is a short simulation of multi lane highway with solid and dashed line markings. Top view of the track is depicted on figure 16.

Figure 16: Top view of the Carla map used for LKA experiment

To be able to compare and visualize the position of the vehicle on the lane and the detection done by sensor, two additional views were prepared. The first view is a top camera view aimed onto the vehicle (so called BEV - Birds Eye View). The second one is a window where detected lines and destination point with the center points are represented on the graph. Figure 17 shows both views from the same simulation time. In the right window of figure 17 all information about LKA status is drawn. Line markings are drawn base on the detected third-degree polynomial and painted with green (right line) and red (left line) colour. Both lines indicate the range view with the intensity of the colour. Type of line is also considered, left one is dashed and right one is solid. Further, there is a red circle showing the position of the calculated destination point and additionally yellow dots, which describe predicted center line. In figure 18 image visible for the sensor, at the exact same moment as in figure 17, is depicted.

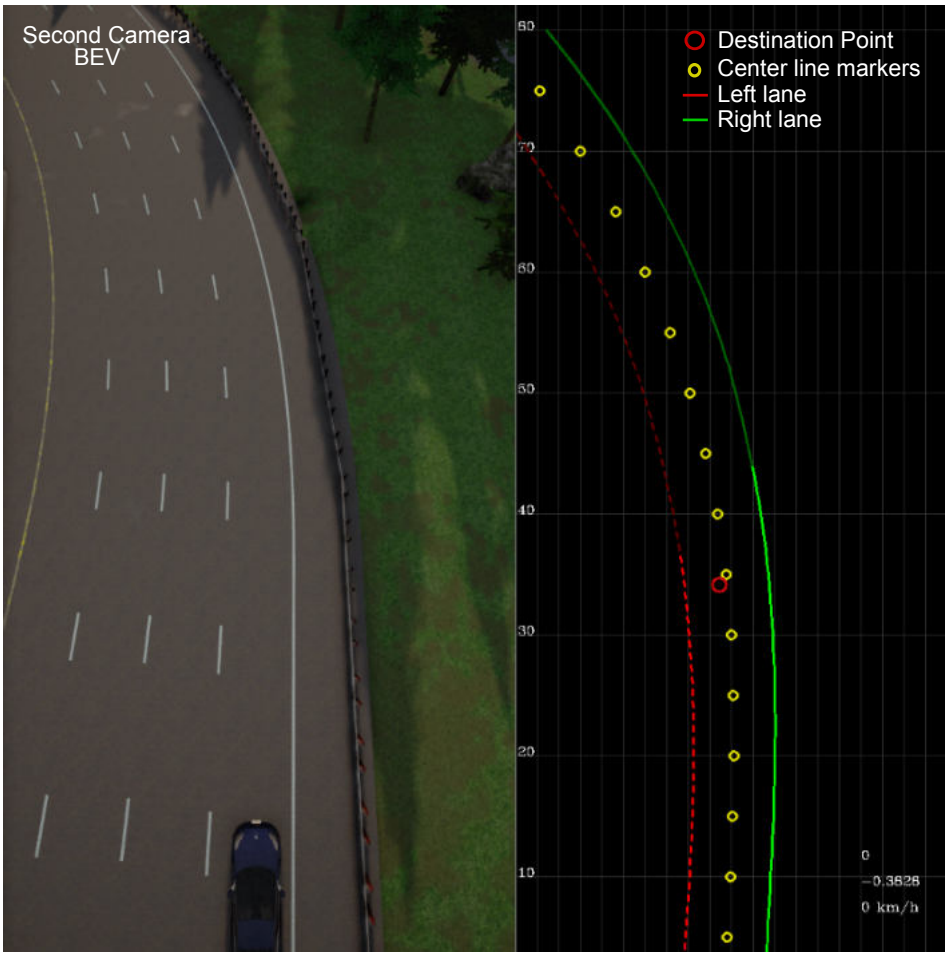

Figure 17: Left: Top BEV camera aimed on the vehicle with visible line markings. Right: visualization of the detected lines with red destination point marked.

Figure 18: Mobileye sensor view of a curve road from the same time as in figure 17.

In figure 17 it is visible how the additional part of the equation 3 were $C_{0\Delta}$ is added, works in practice. Vehicle is slightly closer to the right line, so the destination point due to the added $C_{0\Delta}$ is

a bit closer to the left line. This compensates the center positioning of the vehicle while it moves further on the road. Thanks to the visualizations of detected and road line marking, in figure 17, it is easy to compare the shapes of model and real line spotted by sensor.

Developed LKA function on Mobileye 660 is able to constantly steer Carla vehicle on the endless loop of the test track. In comparison to the first version of the algorithm [40], it does not show oscillations on the straights, even with higher speed over 80 km/h. Nevertheless, there are still problems observed for higher vehicle speeds to properly start steering on the sharper corner, which leads to the drop of detection and fall out of the lane. Those problems are probably caused by the sensor visibility range (80 m) and the sampling of the lane (60 ms). Additionally, there is a lag between steering and the reaction of the vehicle, which is not considered by the algorithm.

Carla simulation includes autopilot, which steers the vehicle to drive around the track. With the usage of this autopilot it is possible to compare the steering input of the vehicle between LKA and Carla autopilot. For this purpose autopilot was run together with Mobileye line detection and LKA. The output with both steering values was logged into the file, so it's possible to plot and compare the results. Figure 19 depicts the plot of LKA and the autopilot steering value on the same track.

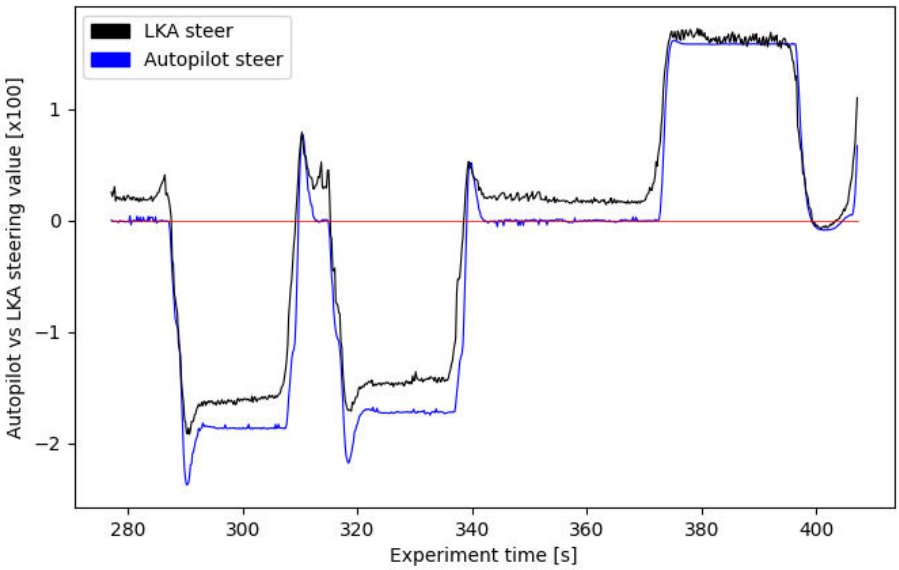

Figure 19: Carla autopilot vs LKA steering values.

Plot in figure 19 is only a part of the track including three corners and two straights. Full run on the track is long and, therefore, in order to show the differences between values, it is better to plot only the part of the run. This is the reason why the plot does not start from zero. Generally, the plot shows that LKA steering values follows the values of autopilot. Short explanation is needed for the values on straight and first two left corners. It looks like LKA keeps turning the vehicle on the straight (between 350 s to 370 s). The explanation of such a situation is depicted in figure 20, where LKA work is plotted on the straight road.

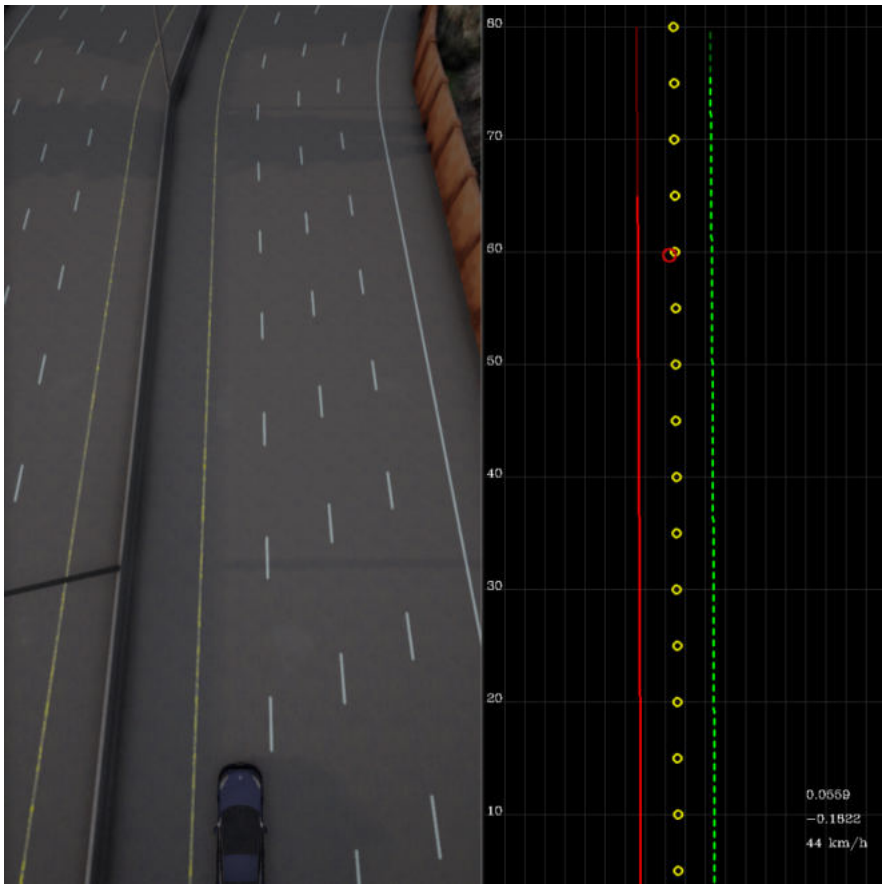

Figure 20: Example LKA work on straight while Carla autopilot controls the vehicle.

Carla autopilot does not drive exactly on the middle of marked lane. Vehicle is closer to the right line than to the left. LKA tries to fix the centering of the vehicle by steering slightly to the left. But since the autopilot drives the vehicle, the difference exists for the duration of the straight. The same issue can happen for the corners. The example of such a drive is visible in figure 19 between 290 s and 310 s. Those two corners are not driven by autopilot fully on the center of the lane. Therefore, LKA is trying to fix it with a bit different steering value.

Experiment results prove that the synthetic data generated lively by the simulation can be used to test and validate the ADAS functionality in the closed loop setup. This type of validation can significantly reduce time of the development of new functions. Moreover, validation of such a function can be moved from the real track validation to the automated closed-loop HIL test.

5 Substitution of the lidar sensor data in real-time application

LiDAR sensor has unique properties, which are not available for other commonly used sensors like camera or radar [17]. Multi-channel LiDARs have rotational core emitting electromagnetic waves, which reflect on the objects and partially return to the sensor. Time of flight of those beams gives the information about object distance. N-channel LiDAR emits laser beam on a specific elevation angle, and the position of the spinning core at the moment of sending the light pulse gives the vertical information about the object. The combination of precise mechanics and electronics allows LiDAR to scan 360 degree view of its surrounding with high precision of distance measurement. The output of the sensor is mostly point cloud, where each point is a reflection of the object, which in visible in the sensor FOV. Newest available LiDARs generate millions of points per second with hundreds of channels. Those properties of LiDAR sensors make them unique and valuable source of spatial information. There are plenty of available LiDARs with different amount of channels and view ranges. That type of sensors has one big disadvantage, which is the cost. High precision mechanics and electronic parts are valuable, which makes the sensor the most expensive, in comparison to camera or radar. That is why LiDARs is mostly used as a ground truth for other sensors or in the research and development test vehicles. Technology is developing, what makes sensors less and less expensive each year. That is why probably every autonomous vehicle will be equipped with a sort of LiDAR sensor in the future. There are already plenty of projects, which relay on the LiDAR sensor [2, 21, 42, 43]. Promising results are achieved in terms of precise object detection based on the LiDAR point clouds. Since there are a lot of researches which successfully substitutes the camera sensor data to improve perception systems [9, 7, 34], the same approach, with synthetic data application can be tried on the LiDAR point cloud data.

5.1 VLP-16 LiDAR sensor simulation

Computer technology called ray-casting is one of the best available ways to generate point cloud out of the 3D rendering simulation. Ray-casting is a computer methodology, which casts virtual light rays onto the objects to render 3D representation of the scenario. This methodology relay on the collision boxes, which may be or not defined for each virtual object. The difference between quality of collision boxes is crucial in terms of proper point cloud simulation [38]. Carla simulator with the native support of the ray-casting delivered with UE4 is able to generate synthetic point cloud out of the road scenario. Highly configurable model of LiDAR sensor available in Carla helps to adopt the outputted point cloud to e.g. mimic the shape and characteristics of the real LiDAR sensor. Nevertheless, sensor mode in Carla is not able to mimic exactly character of the real sensor. The main missing point in this ray-casting method is that is does not simulate the reflective properties of the material. Real sensor, except of the distance to the object, measures also the intensity of the returned wave, which brings the information about the type of material which was hit. This missing point in simulated model makes synthetic LiDAR data less realistic, especially if the considered scenario includes lots of low reflective materials like glass.

Velodyne VLP-16 LiDAR is the popular LiDAR model, which is used by the researches. That is why the next research experiment consisted in simulating the output of the Velodyne VLP-16 LiDAR and using this data in the manufacturer software to render 3D Carla scenario. Available parameters for the Carla LiDAR model will be shown on the VLP-16 example. Sensor model allows to manipulate the FOV, number of channels, rotation speed, number of generated points per second

Table 1: Simulated and real VLP-16 LiDAR parameters.

Parameter	Lidar Simulation	VLP-16
Number of Channels	16	16
Measurement range [m]	100	100
Range Precision [cm]	-	± 3
Vertical FOV [◦]	+15/-15	+15/-15
Horizontal FOV [◦]	360	360
Vertical angular resolution [◦]	-	2
Horizontal angular resolution [◦]	-	0.4
Rotation speed [Hz]	10	10
Point per second [1/s]	144k	144k

and others. All those parameters were set is such a way to mimic the point cloud generated by the real VLP-16 LiDAR. Table 1 lists the parameters of the real VLP-16 LiDAR and the simulated one.

Parameters of the simulated model can be set identically to the real sensor but not all of them can be accessed directly. For example vertical angular resolution results from the number of channels and the vertical FOV. To get the vertical angular resolution, vertical FOV of 30 degree has to be divided by the 16 channels. As the result, the simulated LiDAR will have approximately 2 degrees of vertical angular resolution. Horizontal angular resolution results from the rotation speed, the number of channels and the amount of points generated by the sensor per second. For instance in order to get exactly the same horizontal angular resolution with twice lower speed, the sensor has to generate half of the original number of points per second. If the number of points per second stays at the same level, resolution will be doubled. Formula 8 defines the relation between those parameters.

$$P_{ps} = \frac{FOV_H * N * f}{\phi_H} \tag{8}$$

where:

P_{ps} = Points generated per second [1/s]
FOV_H = Horizon field of view [◦]
N = Number of channels
f = Rotation frequency [Hz]
ϕ_H = Horizontal angular resolution

In terms of the range precision, simulated sensor is perfect. There is no randomness in the distance measurements obtained by the simulation. To generate more realistic point cloud, additional parameters are added to the sensor model. More details about this solution are provided in the next chapter.

Point cloud generated by simulation is described by the set of Cartesian coordinates, expressed in meters, relative to the position of the sensor. It is possible to manipulate the position of the sensor in the simulation. The parameters of the sensor position are relative to the simulation actor, which carries the sensor. Sensor can be placed anywhere around the particular actor, and moves together with it. To visualize how simulated LiDAR data looks like, dedicated image

window was developed in Morphee. Point cloud image is an Open-CV [44] window which renders BEV projection plane of simulated point cloud. Example view of such projection of point cloud data is depicted in figure 21. Similar approach was performed in [45] publication, where BEV Carla projection was also used to compare it with the KITTI dataset BEV point cloud projection. Carla version used by researchers had simplified collision boxes of actors, which made the LiDAR data less realistic.

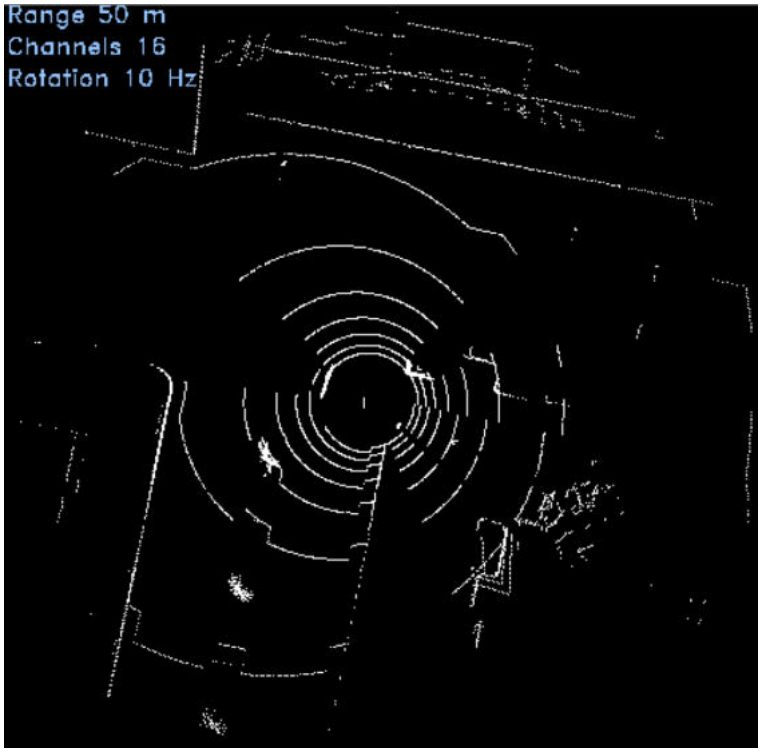

Figure 21: Example BEV projection plane of simulated LiDAR data.

Since BEV projection shows only the top view of the point cloud data, in figure 21 there are only presented the edges of the vehicle and other obstacles visible. It's also visible how perfect are the distance measurements of simulated LiDAR model. Each of the channel which hits flat road surface, creates almost perfect circles.

5.2 Real-time injection of the VLP-16 simulation data for 3D rendering software

The first experiment for the synthetic LiDAR data was to validate those data on the Velodyne manufacturer 3D rendering software. VeloView is a tool created to visualize 3D point cloud generated by the LiDAR sensors. Via Ethernet connection, it can communicate with the hardware of the sensor to receive and draw the output of the sensor. According to the Velodyne documentation, the sensor reshapes the point cloud into the byte stream and over UDP protocol sends each chunk of the scan to the software. Tool decodes the byte stream and renders the 3D view gathered by the sensor. To be able to inject the simulation LiDAR data point cloud, it has to be converted into the same shape of the byte stream as it is done in the real sensor hardware and then send it over UDP

to the VeloView tool. The main difference between the simulated and the real data is that Velodyne sensors provides the point cloud in the Spherical coordinate system. Each point is described by two angles: elevation and azimuth and by the radius to the point. Figure 22 depicts the transformation between Cartesian and Spherical coordinates.

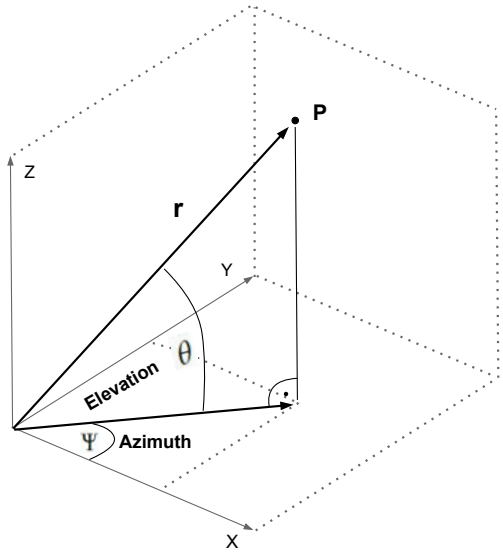

Figure 22: Transformation between Spherical and Cartesian coordinates systems.

Equation 9 defines how to convert X,Y,Z coordinated into Ψ,θ,r.

$$\begin{aligned} \Psi &= arctan(\frac{X}{Y}) \\ \theta &= arctan(\frac{\sqrt{X^2+Y^2}}{Z}) \\ r_s &= \sqrt{(X^2+Y^2+Z^2)} \end{aligned} \tag{9}$$

where:

Ψ = Azimuth angle [rad]
θ = Elevation angle [rad]
r_s = Radius [m]

Conversion to the Spherical coordinates provides direct information about point assignment to the channel, which sent the light pulse. Elevation angle must correspond to one of the sensor channels. That type of conversion creates the matrix where the columns are azimuth angles, while the rows are the number of the channel and the values for each cell is a radius. Table 2 describes how this data matrix look likes.

Table 2: Description of VLP-16 point cloud data flow matrix

	Azimuth N	**Azimuth N+1**	**Azimuth N+x**
Channel 1	r_s1	r_s2	r_sx
Channel 2	r_s1	r_s2	r_sx
Channel ...	...	...	...
Channel 16	r_s1	r_s2	r_sx

Thanks to this type of data stack, it's easy to convert the data into the byte stream. Velodyne protocol defines blocks of byte stream, which include twelve consecutive columns of azimuths. Those blocks are later send one by one to the VeloView tool as UDP frames. Frames are then reshaped into 3D representation of the sensor point cloud. To be able to substitute the real sensor data and inject synthetic one into the VeloView in the real-time, all described conversions must be done on fly by the computer. Dedicated component called LiDAR Simulator was developed in Morphee platform to handle this task. Diagram graph in figure 23 describes the processing of Carla LiDAR data into VeloView 3D render.

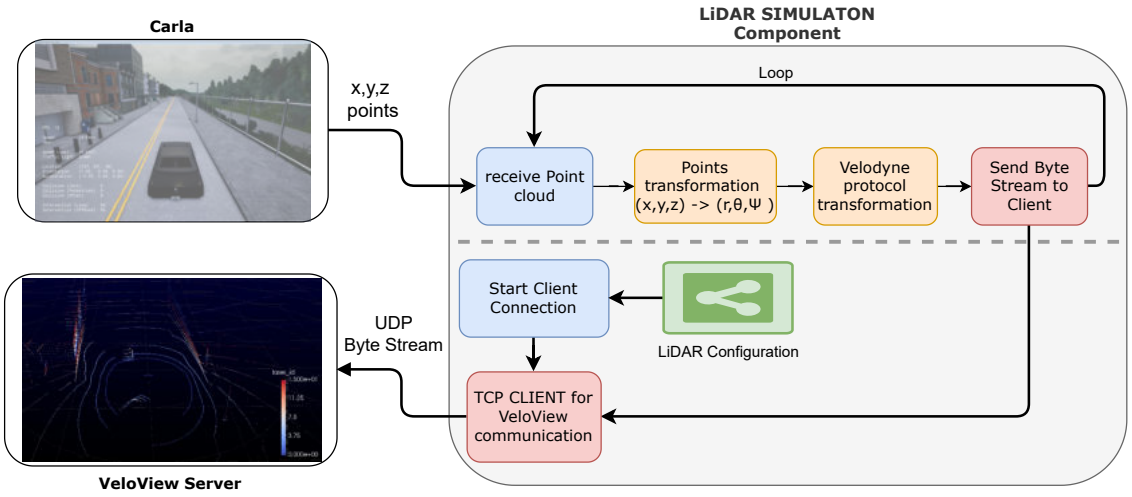

Figure 23: Processing Graph of the Carla LiDAR data into VeloView 3D render.

Raw Carla point cloud is received by the LiDAR simulator component. Next step is to convert Cartesian to Spherical coordinates and, after that, shape the Velodyne byte stream out of those data. At the same time, the other thread starts client and connects to the VeloView server over the local network. Prepared client is ready to receive the byte stream generated by the LiDAR simulator component. Once computed data is available, it's send to the VeloView tool. Correctly shaped and received data renders 3D representation of Carla sensor point cloud in the real-time. For the comparison purposes, figure 24 shows how the real Velodyne VLP-16 3D point cloud looks like. Example LiDAR frames were gathered during one of the test drives of the FEV Smart Vehicle project car [46]. On the proposed view, each channel of the sensor has a dedicated colour. The considered frame depicts the road crossing with visible car driving exactly in front of the test vehicle. Most of the flat road surface is visible on the point cloud as almost perfect circle arc. On the bottom right of the picture there is a fence, which disturbs the light pulses. The points reflected

from the surface behind this fence are much more turbulent. This is an example phenomenon which is not mimicked by the simulation model. None of the simulated object can influence the multiple ray-casting pulses used by simulation to deliver points.

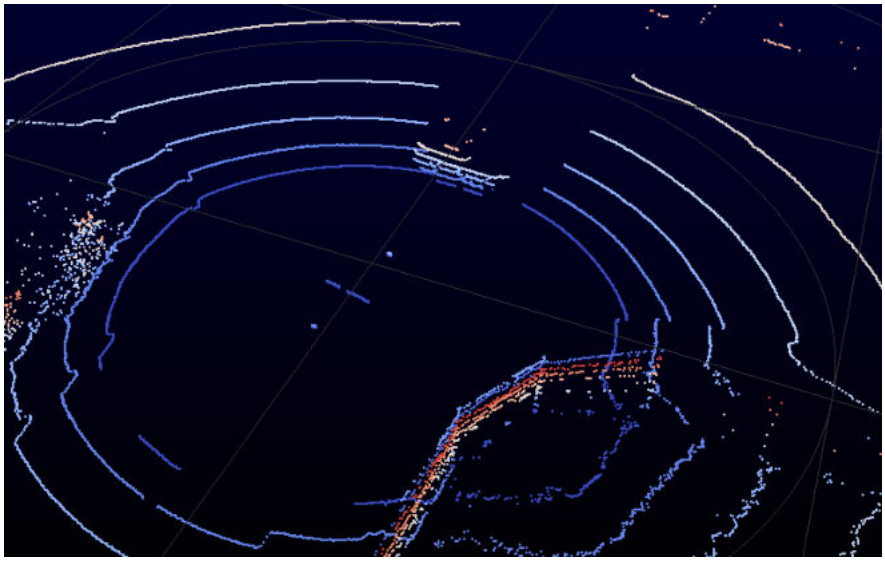

Figure 24: Example VeloView 3D point cloud, generated by real VLP-16 LiDAR.

The successful real-time transformation of LiDAR data allows to render 3D Carla road scenario in VeloView tool. Example scene out of Carla simulator is shown on figure 25.

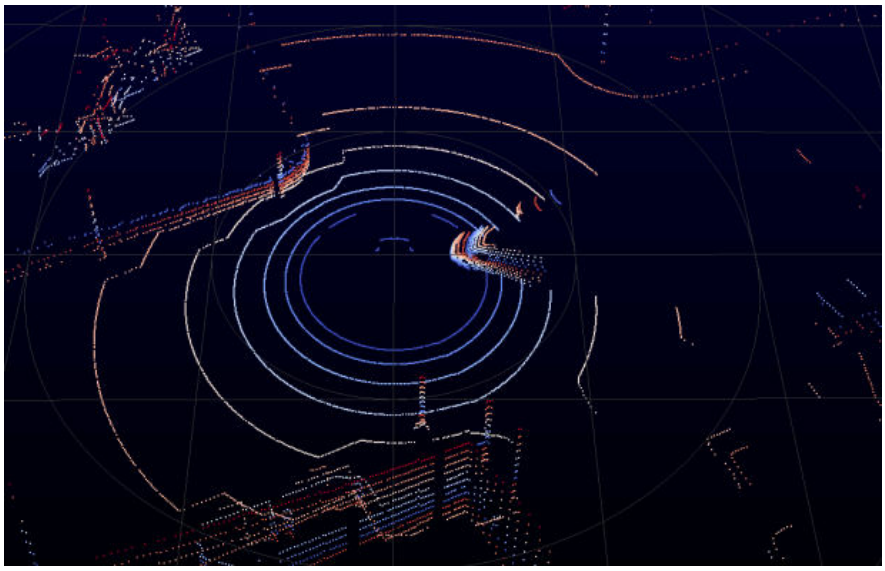

Figure 25: Example VeloView 3D point cloud, generated by Carla LiDAR synthetic data.

All the points gathered by Carla are almost perfect in terms of their position. There are no missing parts of the sensor view, all objects reflect rays exactly the in the same way. Since the solution to render point cloud out of simulation works, now it's possible to explain important additional LiDAR model features.

5.2.1 Improved simulation of LiDAR point cloud

The first important improvement of synthetic point cloud data quality generated by Carla simulation is more precise shape of the collision boxes. There are a few publications which deal with this problem. Researchers in [38] described exactly that the simplified representation of actors, which are visible for the sensor as straight cubes, is totally different than in case of real objects. This problem was fixed in the newer versions of Carla software (higher than 0.9.9). Figure 26, depicts the difference between the shape of the actors scanned by the LiDAR in Carla version 0.9.9 vs version 0.9.12. Newer version, with more precise collision boxes for actors, was used for the further experiments discussed in this thesis.

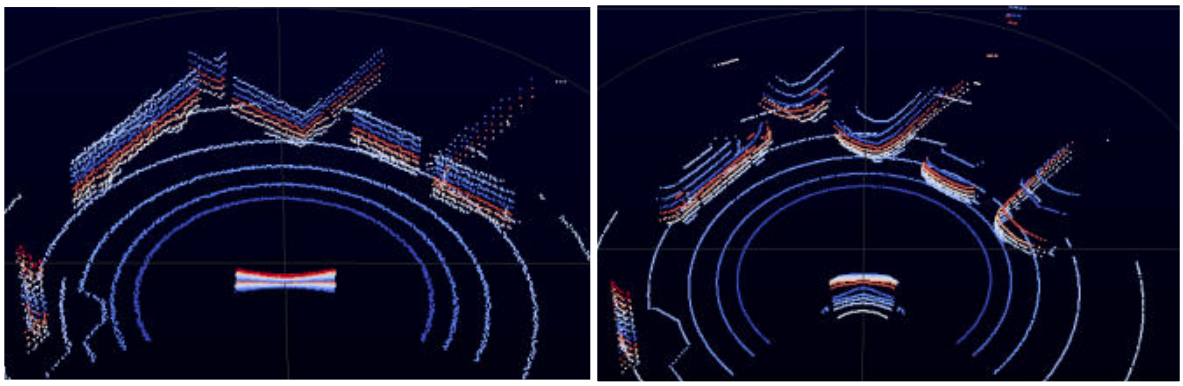

Figure 26: Old vs new models of actors rendered by LiDAR sensor in Carla.

The second important improvement, which is available in the newer versions of Carla simulator, is related to extended parameterization of the LiDAR model. The list of those parameters is defined in table 3.

Experiment of the real-time injection of synthetic point cloud out of Carla simulation resulted in successful live render of 3D scanned simulated road. Generated and reshaped point cloud can be used as the input for the Veloview tool, which was developed for real sensor data. Success of this experiment gives opportunity to use the synthetic data for real ADAS functionality, which relays on the LiDAR data. The goal of this work is to validate the usability of synthetic point cloud in the real applications. That's why further experiments involving synthetic LiDAR data were conducted.

Table 3: Additional sensor model parameters with description.

Parameter	Default value	Description
Atmosphere attenuation rate	0,03	Coefficient that measures the LiDAR intensity loss per meter
Dropoff general rate	0,35	General proportion of points that are randomly dropped
Dropoff intensity limit	0,8	For the intensity based drop-off, the threshold intensity value above which no points are dropped
Dropoff zero intensity	0,4	For the intensity based drop-off, the probability of each point with zero intensity being dropped
Noise standard deviation [m]	0,02	Standard deviation of the noise model to disturb each point along the vector of its raycast

Properly selected parameters from table3 make the LiDAR point cloud more realistic. Those parameters introduce randomness to the process of points gathering. Noise standard deviation can add realistic error to the distance measurements, like real LiDAR does. Some of the parameters relay on the simple implementation of the intensity model. For each point, intensity is calculated based on the travel distance. Equation 10 defines how it is calculated.

$$I/I_0 = e^{-a*r_s} \tag{10}$$

where:

I/I_0 = Intensity

a = Atmosphere attenuation rate

With those additional parameters randomized point cloud looks more realistic than the perfect one generated by simulation. To be able to compare the difference between perfect and modified point cloud figure 27 was created.

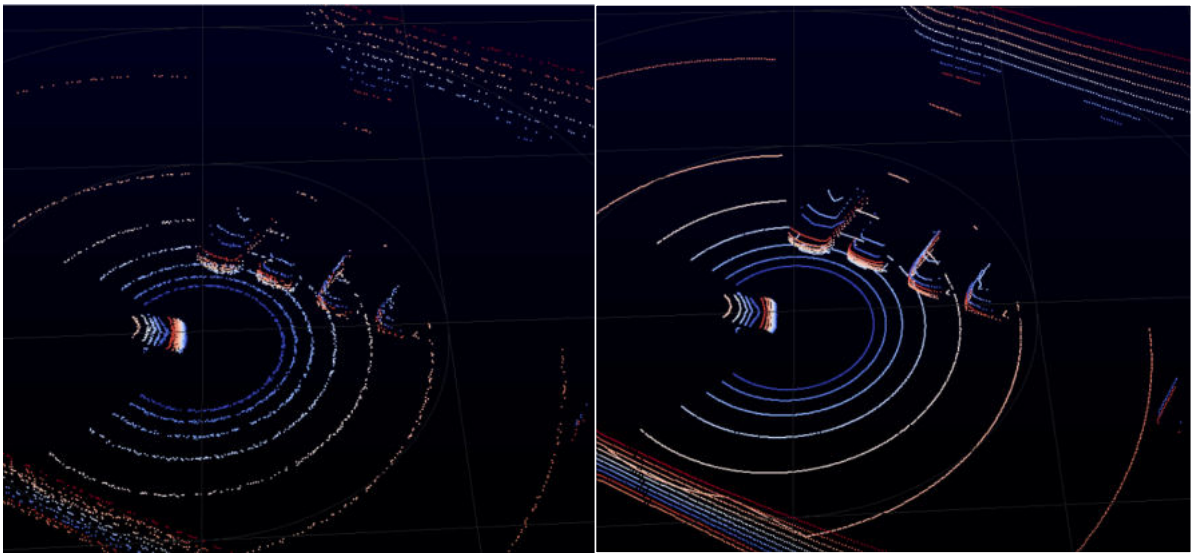

Figure 27: Comparison of point clouds, with and without additional parameters set for the simulation model.

Both images included in figure 27 depict the same road situation. There are four cars standing next to each other and ego vehicle head to them. On the right side of the figure, all the points are rendered and the edges of the objects are sharp. In case of the left image, where additional model parameters are set, the scenario representation is less sharp and more grainy. Proper set of additional parameters allows to mimic the real point cloud data more realistically. But as already mentioned, without proper implementation of intensity for the material in simulation, point cloud won't look exactly the same as the real one.

5.3 Lidar free space detection comparison on real and synthetic data

In the FEV company project called Smart Vehicle [46], the VLP-16 LiDAR sensor was used for ADAS research experiments. One of the functionalities developed in this project was a free space detection algorithm. The goal of this algorithm is to determine which part of the road in front of the vehicle is free and which is occupied by some obstacles. Algorithm was classically developed without the usage of machine learning technology and only with the application of the real sensor data. The vehicle equipped with the VLP-16 LiDAR on its front was used as the source of the data. Figure 28 depicts the image of one of the Smart Vehicle cars used in the project, which has VLP-16 LiDAR attached to the front bumper.

Figure 28: Smart Vehicle car equipped with Velodyne VLP-16 sensor.

Assumed position of the sensor in the test vehicle decreases the vertical view range of the sensor to the front 180 degrees. Rear half of generated points is useless, since all are blocked by the car. Implemented algorithm can determine free or occupied spots of road only in front of the vehicle. Sensor can generate the data but computation has to be established on the dedicated hardware. To avoid the necessity of sending all point cloud data from the vehicle to the computer via wireless connection, dedicated hardware was fitted into the car trunk. For this purpose the Nvidia Drive PX platform has been selected. This relatively small computer delivers high performance with GPU computing abilities. It's highly valuable in case of processing vast amounts of data, for example LiDAR point clouds. Vehicle setup included sensor connected via Ethernet with Nvidia Drive PX platform, which was receiving the point clouds and computing the free space detection. In the next step, the 3D sensor point cloud was rendered in ROS, similarly to VeloView. Correctness of the algorithm performance can be verified on the basis of the graphical representation (the contour indicating if given part of the road is free or occupied by some object) provided by the algorithm. Figure 29 depicts two example scenarios where free space detection is drawn.

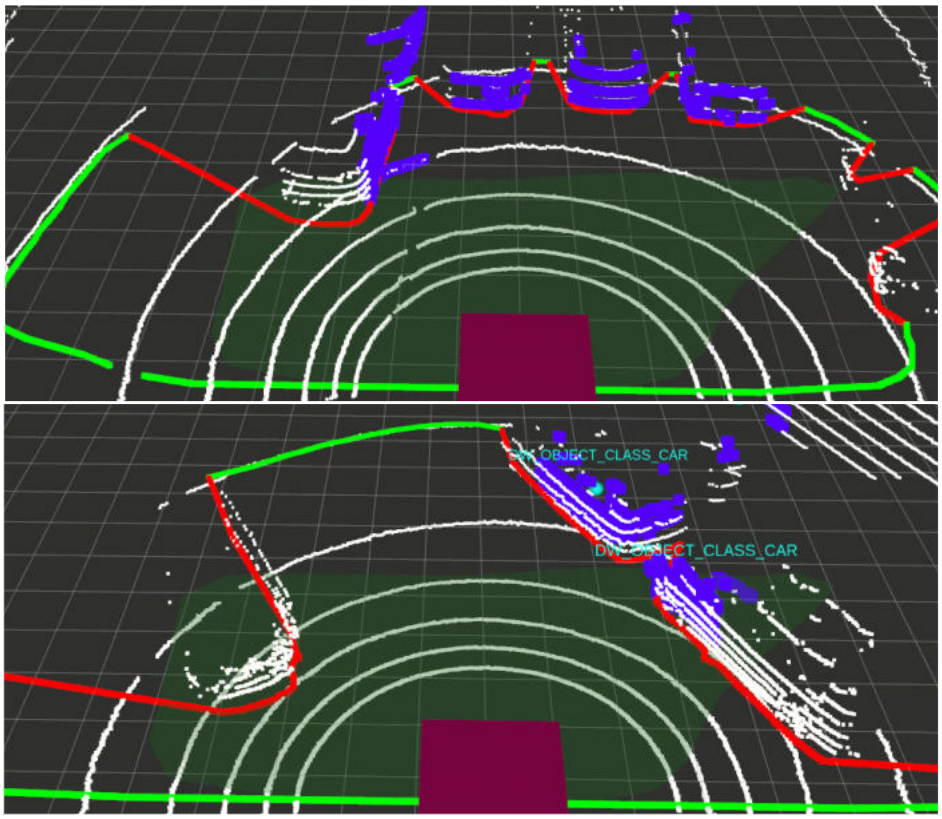

Figure 29: Two example images, where free space detection algorithm works on real VLP-16 LiDAR data.

Purple boxes at the bottom center of the images 29 determine the positions of the test vehicle, which is equipped with the sensor. Points of sensor data are white and the contour generated by the algorithm is marked with green and red lines. All road spaces which finish with green contour are detected as free and those which are occupied are marked in red. On both examples there are cars standing on the ego vehicle road patch. 16 channel LiDAR does not deliver dense enough point cloud to detect correctly obstacles far from the sensor. Therefore the algorithm works at a short range in front of the vehicle only.

5.3.1 Free space detection on synthetic LiDAR data

The real system including the vehicle and hardware defined how the free space detection algorithm works in real applications. Now the goal is to experiment if synthetic LiDAR data are able to substitute the real sensor data within the system and deliver comparable results. Is it possible to run the same algorithm, which was developed and tested only on real LiDAR data, on Carla simulated data? Previously conducted experiments with the live injection of VLP-16 LiDAR sensor data provided straightforward core to use it with free space detector run on the vehicle hardware. The first step was to develop the validation system including the same hardware parts as in the test vehicle and substitute only the sensor data input. Figure 30 shows the graph of the developed real LiDAR sensor substitution with the usage of Carla and Morphee HIL system.

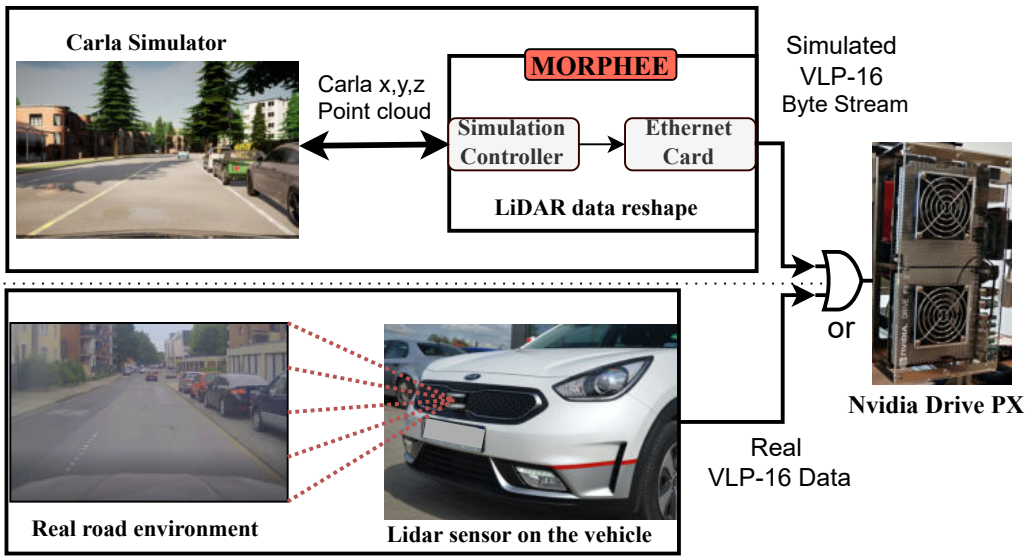

Figure 30: Graph explain how real VLP-16 data is substituted by synthetic Carla sensor point cloud.

Experiment discussed in the previous subchapter 5.2 presented how point cloud data is processed and send to the VeloView tool. In this experiment the idea is to use the same processing and sending method to deliver it into the Nvidia Drive PX hardware. There data can be handled the same as a real sensor data. If we simply unplug the Ethernet cable from the sensor and plug the other one from computer where simulation generates the sensor data, then, from the perspective of the Drive PX hardware, there is no change. Data flow is the same as in real application. To be able to visually compare the free space detection algorithm which works on the synthetic data, similar road scenario as on the real example was prepared in simulation. Parking situation from the top image in figure 29 was created. The scenario included three lined cars in front of the ego vehicle and two standing closer on the sides of the sensor field of view. Figure 31 depicts the camera view of similar scenario prepared in simulation.

Figure 31: Camera view of prepared parking scenario for the free space detection algorithm comparison.

Created scenario, as in real example, included parked vehicle in front and on the side of the ego vehicle. Next figure 32 shows both outputs of free space detection performed with the application of the VLP-16 LiDAR calculated for the real and synthetic data.

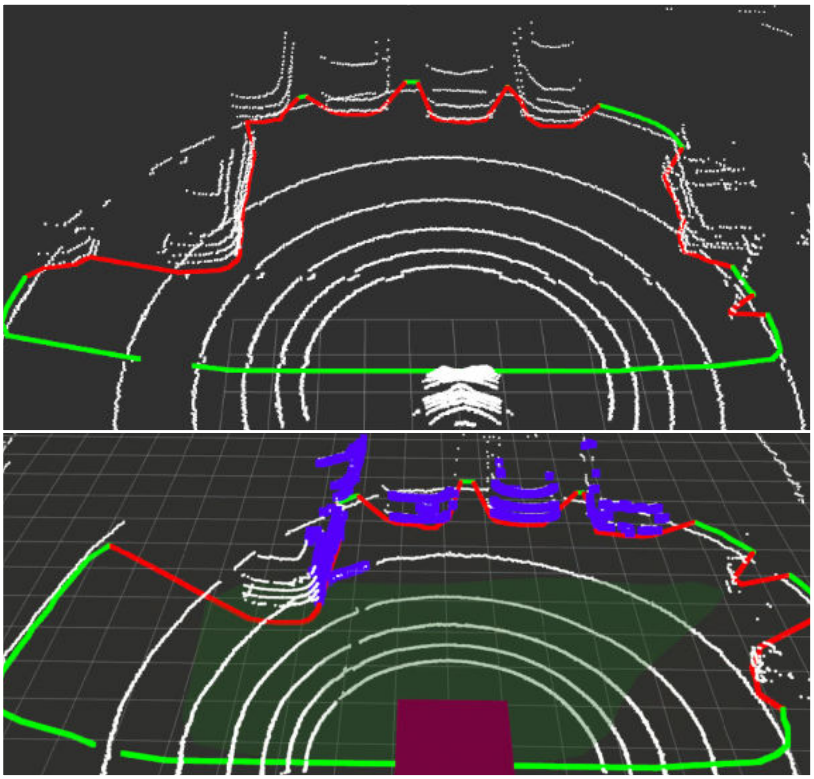

Figure 32: Comparison of free space detection algorithm calculated on real and synthetic data out of similar road scenario.

In the examples created out of the similar scenarios, it is visible that all the vehicles in the sensor field of view are detected and marked correctly with the red edge line. There are no differences between algorithm output generated out of the real and synthetic data. Highly detailed models of objects and random noise added to the simulated sensor data, result in almost identical character of point cloud and the one generated by the real Velodyne VLP-16 sensor. A few more examples were captured to show how algorithm works on different road scenarios. For better understanding of the scenario, the camera image was captured in exactly the same time as in case of the LiDAR point cloud. First examples included imitation of a highway with multiple lanes. Except of the ego vehicle with LiDAR sensor there were two more cars driving in the same direction 33. Both were correctly detected and marked in red by the free space detection algorithm.

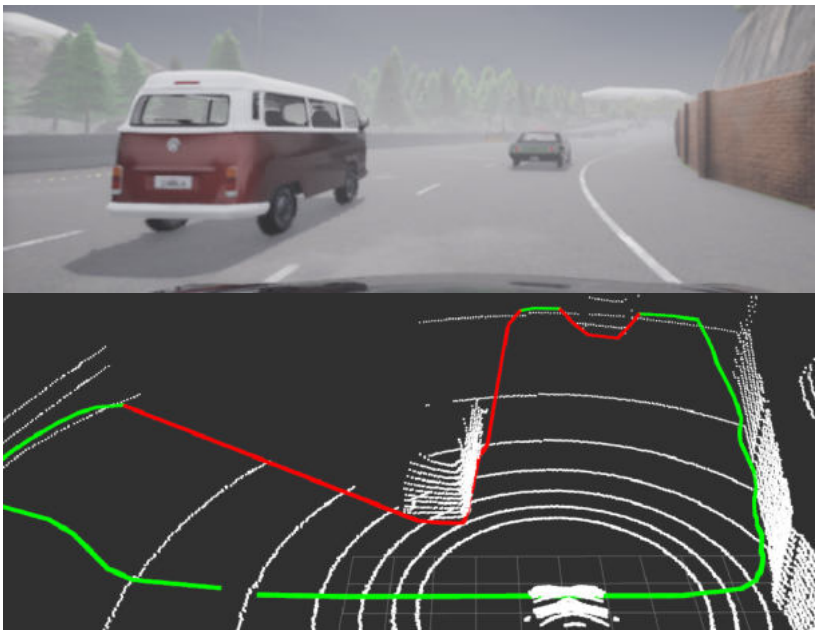

Figure 33: High way example scenario with two vehicles detected by free space algorithm.

Second example included four pedestrians, which were crossing the road. Ego vehicle was stopped just in front of the pedestrians 34. All four obstacles were correctly detected by the algorithm.

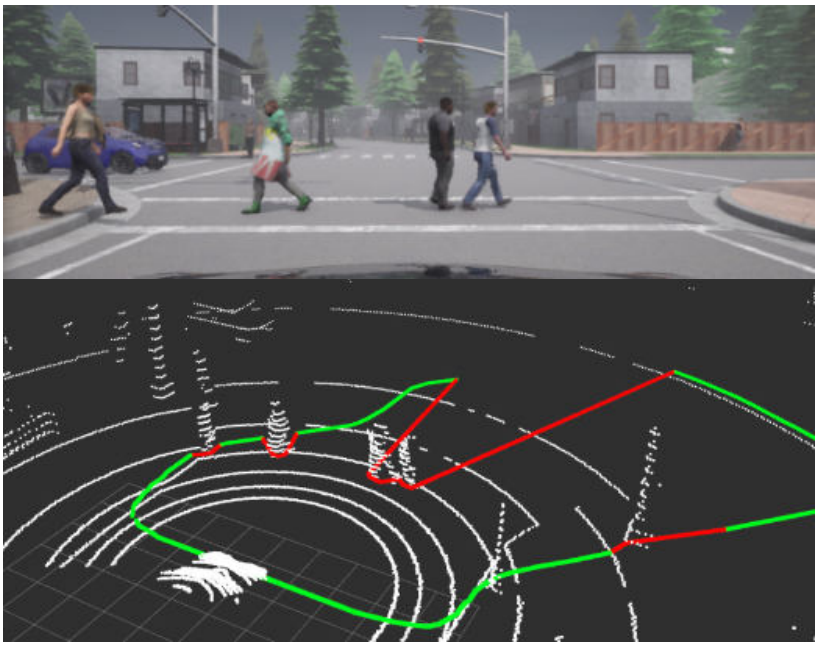

Figure 34: Example scenario where pedestrians are crossing road. All four correctly detected by free space algorithm.

Last example depicts the work of the algorithm on the gas station were one vehicle is parked.

On this example, not only parked vehicle was properly marked, but also one of the pillars of the gas station roof 35.

Figure 35: Gas station scenario, where station pillar and parked car is correctly marked.

All shown examples prove that for the Nvidia Drive PX platform, there is no differences between free space detection obtained on the real or synthetic data. Algorithm developed with the usage of real data only works correctly also on the simulated data. In case of the further development and improvement of the free space detection algorithm in specific road scenarios there is no issue to do this on virtual Carla world. This approach may significantly reduce the cost of development of such ADAS functions.

An interesting conclusion can be drawn from the achieved results. Algorithms or functions developed without supervised learning technology are less sensitive to the character and quality of the input data. Tested in this experiment free space detector is an classical hard-coded algorithm which has no issues to work on synthetically generated data.

6 Study on synthetic LiDAR sensor data usage for pedestrian detector training

The most promising results, considering perception systems for autonomous systems, are delivered by algorithms which relay on machine learning technology. Properly developed, deep neural networks trained with huge amount of labeled input data, detect objects of interest with high accuracy and precision. The main work is required to provide such a vast amount of input data. The effort is not only to record but mostly to precisely label the recordings. Regarding those needs, technology which may reduce effort of data preparation for machine learning is in great demand. Synthetic data generation maybe the solution. That's why the next experiments, where the direct comparison of the pedestrian detectors trained on the real and synthetic data was performed, are the most important in this work. Obtained results may provide the answer to the question, if it is possible to generate such high quality synthetic data to substitute real training data and still get the same or better performance of the algorithm. There are already researches, where this topic was tackled [7, 9, 34] but in case of data form the RGB camera sensor. That's why the focus of this research is to tackle the feasibility study of synthetic data but, this time, generated by LiDAR sensors. In most of the published papers the researchers had failed to bridge the gap between synthetic LiDAR point cloud and the real one. One good example is the publication of Dworak et al. [38], where model trained on the synthetic Carla point cloud performed much worse than the one trained on real point cloud.

The experiment plan for this research was divided into a few consecutive parts. The first step was to select and define the most comparative machine learning model for the further training experiments. Second one was to define the point cloud processing method for the chosen machine learning model. The next step consisted in determining, if the synthetic point cloud from Carla is usable as a training data for the deep neural network. The goal was to determine if the chosen network model was able to learn the shape of pedestrian out of the point cloud and correctly mark it on the input data. The next step was to prepare the valid real sensor dataset for the comparison purposes. Base on those data training abilities, the reference for synthetic data was delivered. The same algorithms and methods used for the real data had to be applied to the synthetic data to perform proper comparison. It should be stressed that since the real sensor data is the input for the ADAS applications, all detectors trained on synthetic data has to be validated on those real data. Otherwise, it may happen that the highly precise model trained and validated on the synthetic data has a very low performance in real applications [38]. The next step consisted in delivering as good as it was possible input data, which could be treated in the same way as the real data selected for validation. The whole process and methods were delivered to mimic the real dataset. The last and most important part of experiments was to play with synthetic and real dataset size to compare the performance of trained detectors and determine the usability of the synthetic data.

6.1 YOLOv4: deep neural network model

First of all, in order to build a proper environment for comparison of performance of model trained on different datasets, the goal for the training has to be defined. For the purposes of synthetic data experiments, detection of pedestrians was selected as the goal for neural network. Pedestrians are the most vulnerable and valuable traffic participants. In comparison to the cars they are smaller,

so correct detection is more challenging. The goal is set, now the input and output for the network has to be defined, before particular model will be selected for the task. LiDAR sensor produces 3D point cloud, so the standard approach is to use 3D input for the neural network model. This is however much more difficulty and, what's more important, there are much less tools and models for training 3D models. That is why the conversion of input data was proposed for this experiment to decrees input data complexity from 3D to 2D, without losing valuable scanned scenario information. This conversion will be precisely described in next chapter of this thesis. Since the input is defined as 2D, the output will be also 2D. The goal is to detect pedestrians, while the output of neural network has the form of positions of detected objects on input data. The array of values which describe unambiguous position of object is called label. What's important, there is no limitation how many pedestrians may be visible on the scanned scene, there may be no pedestrians as well.

The goal is to reach the highest performance of pedestrian detector as possible [47]. That's why, the state of the art deep neural network model was chosen for the experiments. One of the first successful implementations of convolutional neural networks CNN was described by Girshick et al. [48] in 2014. Proposed CNNs were trained to perform accurate object detection and semantic segmentation. Deep learning detection methods are divided into two main categories: two-stage processing and one-stage processing methods. Two-stage processing methods firstly look for the potential boxes, where objects are located and then, those boxes, are processed further to determine the prediction. In case of the second approach, single-stage processing methods deliver object feature map and the final results directly. The first single-stage YOLO (You Only Look Once) detector was proposed in 2015 [49]. This model was treated as a milestone in the computer vision due to its simplicity and high-speed detection capabilities. Since 2015 a few improved versions of YOLO algorithm: YOLOv2 [50] and YOLOv3 [51] were developed. Each has greater detection accuracy in comparison to the first version of the model. YOLOv4 [22] was proposed by Bachokovskiy in 2020. This model provides balance between the real-time performance and high accuracy. What's more, it is characterized by the excellent learning capabilities. This model is a promising choice for the real application for autonomous systems for vehicles and robotics [52]. YOLOv4 model was chosen for the purposes of pedestrian detection experiments as one of the best and popular solutions. To explain the basics of the YOLOv4 model, figure 36 was provided.

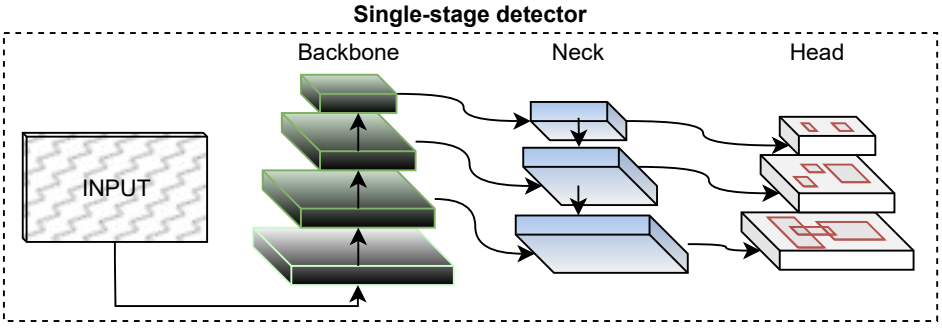

Figure 36: Simplified block diagram of YOLOv4 model architecture.

Figure 36 depicts the simple graph of the YOLOv4 architecture [53]. Presented graph defines three most important blocks of the model. First is the Backbone. Backbone is deep learning architecture, which extracts features from the input. A few different classification models may be

used as a backbone for YOLOv4, for example VGG16 or SqueezeNet. Second block of the model architecture is so called Neck. Neck collects feature maps from different stages of the backbone. The last part is the Head, which is known as object detector. This part of the model is responsible for finding the regions where the objects might be present.

6.1.1 Evaluation metrics

Performance evaluation of any algorithm is crucial in terms of understanding how good is the developed model. The same evaluation process is necessary to compare performance of different algorithms. There are many evaluation metrics used to measure the quality of deep neural network detectors [54]. In the conducted experiments, the output of the algorithm is the object label box, which is the surrounding of the pedestrian visible on the input image. Such an output can be evaluated in two steps. First step is to determine whether the output label box is positioned on the pedestrian. Second one is to calculate how accurate is the position of the box on the pedestrian. The most common metrices for this kind of evaluation are based on the number of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) in a confusion matrix [55, 56]. To define the meaning of those categories, an additional variable: Intersection over Union (IoU) must be introduced. The classification of the model prediction, TP, FP, TN or FN is done on the basis of the value of IoU. IoU is defined by the equation 11..

$$IoU = \frac{areaofoverlap}{areaofunion} \tag{11}$$

The meaning of area of union and area of overlap is explained graphically in figure 37. IoU is an value from zero to one, which gives direct information how much overlapping exists between prediction bounding box and the validation one, from the test data.

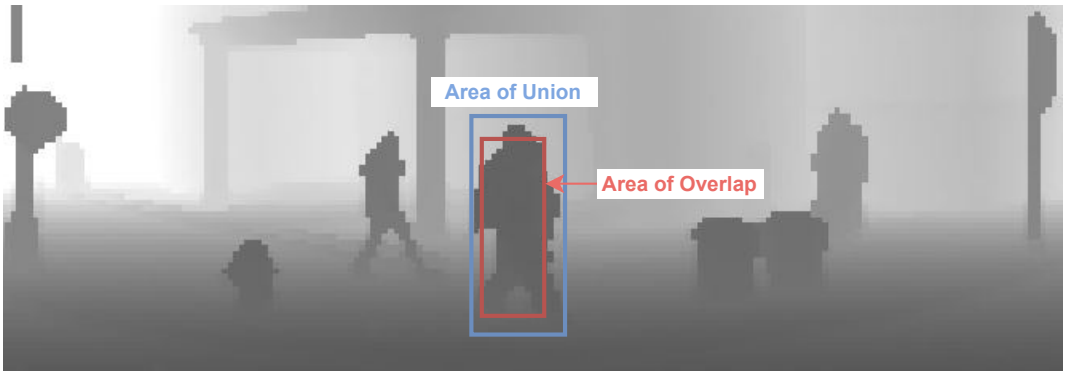

Figure 37: Example shows how the area of overlap and the union is defined for a given object.

On the basis of the IoU value it is possible to determine how to classify the prediction:

- True Positive: correct detection. IoU $\geq$ threshold.
- False Positive: mismatched detection on existing object. IoU $<$ threshold.
- False Negative: no detection on existing object.

- True Negative: correct no detection.

In most practical applications the threshold for IoU is set to 0,5 value [57]. In some cases, as for this research, value of IoU might be different. It helps to understand the differences and weak points of the trained models. It is important to mention that the deep neural network output provides also one additional information, which is detection confidence. This variable informs how confident is the algorithm that a given object exists in a particular area. The confidence threshold may impact to the amount of TP and FN. For some confidence thresholds a label may be treat as valid and pass the IoU test and for other the same label may be treat as unimportant and, as such, generate additional false negatives. The last and most informative evaluation metrices can be calculated based on the amount of TP, FP and FN. Those metrics are called Precision, Recall and F1-score. All are defined based on TP, FP and FN. Equation 12 defines how to calculate Precision, Recall and F1-score.

$$\begin{aligned} precision &= \frac{TP}{TP + FP} \\ recall &= \frac{TP}{TP + FN} \\ F1 - score &= 2 * \frac{recall * precision}{recall + precision} \end{aligned} \tag{12}$$

The training environment calculates also the value of mean average precision (mAP). This metric is based on the Precision and Recall evaluated for each class, which is calculated out of the model validation. For a particular detection class an average precision is calculated. From the definition, estimation of an Average Precision (AP) consists in finding the area under the precision-recall curve. Equation 13 shows the mathematical definition of AP.

$$AP = \int_0^1 p_r(r_c)\, dr_c \tag{13}$$

where:
AP = Average precision
p_r = precision
r_c = recall

Both values AP and F1-score can be used to define the performance of detection model. The main difference is that F1-score value indicates the score for the certain confidence threshold and AP gives more overall view of the model stability across the different confidence thresholds. In the course of this research, both metrics F1-score and AP will be used to compare and evaluate the pedestrian detectors performance.

6.1.2 Darknet - AI framework

In order to carry out any kind of deep neural network training, high performance tool-chain is needed. There are multiple open source frameworks to perform AI computations [61]. One of the most popular frameworks, developed by Google, is TensorFlow. It is used by many researchers and student all over the world. Its simplified library called Keras, makes it even more easy to

use by almost everyone. Most of frameworks are runnable by Python scripts, which is the most easy and fast programming language. PyTorch, Spark MLlib, Theano or Scikit Learn are the other Python based frameworks for AI computation. For this researcher the most important factors for the training task were the speed and performance of the training. Since the Python is not the fastest programming language, C++ based framework was selected. The second important factor for the choice of the AI framework was to support the GPU accelerated learning process. The simple neural network models can be trained with CPU but those really complex require much more computation power, which can be delivered by GPU cards. So called *Darknet* framework [62], written in C++, was selected for the LiDAR object detection with the application of the YOLOv4 model. Darknet was developed by Joseph Redmon, which is the first who work on YOLO algorithm. This framework is one of the best to select for training YOLOv4 architecture. Darknet is a powerful tool-chain, which uses Nvidia CUDA drivers (Compute Unified Device Architecture), to work directly with the GPU virtual instruction set. Framework can run the same training process over multiple GPU cards installed in the PC. The initial experiments with YOLOv4 in this research were carried out with a single GPU (Nvidia Geforce GTX 960), while the last with two Nvidia Geforce GTX 1080Ti. Even with the usage of both of those graphic cards, single fine-tuning training took around 2-3 hours. For all the experiments plenty of training loops were performed, for different datasets and hyper parameters of the model.

Except of the Darknet environment, the popular TensorFlow 2.6.0 framework was used for this research. Darknet was strictly used for the training process while the TensorFlow was necessary to read Waymo dataset. All recordings of Waymo data are stored in the format of *.tfrecord*. To be able to read out data out of those files, Python scripts withing PyCharm program were developed. More details concerning this process are provided in subchapter 6.4, where preparation of real LiDAR data for validation purposes is described.

6.2 Range map processing out of the LiDAR point cloud data

Most of LiDAR sensor approaches create detection models to work directly on 3D point cloud generated by sensor. This way is the most natural, while working with 3D data. One example was already described in the previous chapter, where a free space detection was calculated directly from the sensor point cloud. There are already published works, where 3D LiDAR data was used to train the deep neural networks [35, 38, 58, 59]. 3D point cloud gives direct spatial information about scanned scenario, but in comparison to 2D images, it is much more complex and slower. The additional dimension comparing to 2D projections makes deep neural network architecture much bigger and complex. Due to those reasons, a few different 2D projections of LiDAR point cloud are used to detect objects from the sensor data. BEV (birds eye view) is a type of projection from the top of the sensor, which shows the xy plane of the scenario [60]. In this type of projection the height of the objects is lost, due to flattening of the data onto xy plane. Similar to this idea, in case of the front view (FV), the projection of the yz plane of the point cloud is considered. Here the depth of the objects is lost due to projection. There are also approaches where multiple 2D projections are set as matrix and used as the input for the algorithms. Those are for example Voxel-based or Pillar-based [58] methods. In this case all spatial information is considered, but input matrix of 2D data makes model more complex and slower. In this research, the range image projection [21] is proposed as the best solution, comparing the full data information and simplicity of 2D data. Range image projection may be compared to FV projection combined with depth map. Depth map is an

image, which indicates the distance to the object (in a grayscale) for each pixel. Closer object is more black and further is more white. Example depth map generated in Carla is depicted in figure 38.

Figure 38: Example depth map generated by Carla simulator [63].

The same processing might be performed on 3D point cloud. The position of each point is projected onto front view and the black/white color value indicates the distance to the point. To produce such a front view projection it is necessary to know how to fit x, y, z into range image width and height. The algorithm to produce a range image is pretty much the same as the one already defined for VeloView tool real-time injection (chapter 5.2). The first step is to convert Cartesian to Spherical coordinates. This is defined by equation 9. As the result of conversion, from each point, the vector of three values is obtained. The first value is the azimuth angle. This angle gives direct information about horizontal position of the point in the sensor field of view. The second one is an elevation angle. Each may be assigned to the LiDAR channel giving vertical position of the point. The last one is radius. Radius provides direct information about the distance to the object. This can be turned into each pixel saturation from black to white. To turn radius to pixel saturation, the maximum range of distance measurements has to be considered against the maximum value of the pixel saturation. Formula 14 defines how to calculate the factor of grayscale value to meters. As an example the calculation is done for single channel 8 bit color image.

$$\begin{aligned} G_m &= \frac{max(r_s)}{max(g)} \\ G_m &= \frac{25m}{255} = 0,98[\frac{1}{m}] \end{aligned} \tag{14}$$

where:
G_m = Grayscale factor per meter [1/m]
g = grayscale value

It is possible to define more accurate distance factor for the range map. For example colored image may be coded into point distance. Each color channel R, G, B decodes the distance, from less to more significant bytes. Each channel is still 8 bit, but there are three of them, so the maximum number of possible values is 255 to the power of 3. Formula 15, is an example of such 3 byte coding, used in Carla software for depth map distance calculation.

$$\begin{aligned} G_m &= max(R) * \frac{R_1 + G_2 * 256 + B_3 * 256 * 256}{256^3 - 1} \\ G_m &= 1000\frac{1}{256^3 - 1} = 5,96 * 10^{-8}[\frac{1}{m}] \end{aligned} \tag{15}$$

where:
R_1, G_2, B_3 = Byte value of each color channel

The resolution of such notation, for distance of each point, is much higher than the resolution of any LiDAR sensor.

To be able to prepare an example of range image, it is necessary to define a few LiDAR parameters. For the first experiment with synthetic LiDAR data training, the Velodyne HDL-64E was chosen. This sensor is widely used by the researchers just after VLP-16, due to much higher number of channels - 64 [64]. Sensor parameters are defined in table 4.

Table 4: HDL-64E LiDAR parameters.

Parameter	**HDL-64**
Number of Channels	64
Measurement range [m]	120
Range accuracy [cm]	Up to ± 2
Vertical FOV [◦]	+2/-24.9
Horizontal FOV [◦]	360
Vertical angular resolution [◦]	0.4
Horizontal angular resolution [◦]	0.08 - 0.035
Rotation speed [Hz]	5 - 20
Point per second [1/s]	1.3M

LiDAR model simulated in Carla was set similarly to HDL-64E, to mimic the point cloud produced by this sensor. Horizontal angular resolution is the parameter that depends on the rotational speed. That's why it is defined as a range. Speed to ray cast is the same for all rotational speeds, so lower speed gives higher horizontal resolution.

Sensor parameters are known, so let's define the width and height of the range image. Horizontal field of view is 360 degrees and horizontal resolution is 0,15 degrees. This horizontal resolution results from the 10 Hz rotational speed set for the LiDAR model in simulation. Dividing the horizontal field of view by resolution, gives the image width, which equals 2400 pixels. The height of the image is equal to the number of channels, so 64. Such projection outputs very wide images, with aspect ratio 37:1. To reduce the width of image, horizontal field of view is truncated to front 180 degree. The output image gives 90 degrees of view to the left and right of the sensor. The example image of 1200 × 64 pixels, generated by range image projection out of simulated Carla LiDAR is depicted in figure 39. It is important to mark that for this research the maximum range for LiDAR simulation was set to 25 m. Objects like pedestrians, which are further than 25 m are so small on the image that it is impossible to detected them.

Figure 39: Raw Range image generated out of Carla simulation LiDAR sensor.

On the example range image 39 it is very hard to spot the pedestrian. The grainy character

of the image makes it really hard to understand, what is happening on the scenario. The image is still wide, aspect ratio is 19:1. The grainy character of the image results from the missing points in the input point cloud. The image matrix depicts all possible pixels, so if there is a lack of the data provided by the sensor, white spot without distance information appears. To improve the quality of those images, two additional processing methods were used. First of all, the width of the image was decreased by doubling the value of the horizontal resolution. It resulted in two near 64 pixel columns merging into one column. As the result, the image width was changed to 600 pixels. The same scenario as in figure 39 is shown in figure 40, but after the first step of post processing.

Figure 40: Range image generated out of Carla simulation of LiDAR sensor after the first post processing change.

The image starts to be more clear. Pedestrian standing just on the left side of the image center can be spotted. Still the image is grainy and lots of white points are generated on the picture. To overcome this grainy style of image, the second post processing method was used. Median blur filter with 3 pixels of radius was set on the range image. This filter, removed most of the white spots from the image. The same example after the application of both post processing methods is depicted in figure 41.

Figure 41: Range image generated out of the Carla simulation of LiDAR sensor, after the application of both post processing methods.

Now after all the changes, the range image looks more like black and white camera image, with a bit wide field of view. The wall next to the sidewalk with a few pole like shapes is visible. Even really small curb, which splits the road and the sidewalk, started to be visible. Application of the discussed type of filtering is a bit risky task. With too high radius of the filter it is possible to remove a lot of important information from the input data. A good example is the pole next to the pedestrian. Event with a smallest possible radius for the median blur filter, it stopped to be continuous, which of course is not possible in reality.

Range image projection transformed a 3D LiDAR point cloud into a 2D image representation of the road scenario with precise distance information. It can be compared to the out of the box depth map for the camera sensor with just a bit wider field of view. There is one additional and important advantage comparing to the camera image. LiDAR sensor works perfectly also at night or in very low light conditions. This is something unachievable for the camera sensor. The main disadvantage of the LiDAR application is much lower vertical resolution than in case of standard cameras.

6.3 Generation of synthetic Lidar data for supervised learning

The first step before actual data generation out of simulation, is to define and develop the road scenarios. The goal of the study is to train pedestrian detector based on the point cloud. Prepared precarious should include variety of pedestrians visible for the sensor. The main challenge is to generate diverse datasets. Pedestrians shall be scanned from all the directions, in different poses, standing or walking. There is one concern regarding the available poses for simulated pedestrians. Carla delivers only animation of walking or standing pedestrians and it is not possible to animate the squatting or sitting people. This is of course possible to do it within Unreal Engine 4, but it requires a lot of modifications of the Carla core. This issue may be negligible, since most of people visible on the roads are standing or walking. So the problem of different poses of pedestrians should not have a significant influence on the detector predictions.

The development of the scenarios starts from the definition of the actors positions (ego vehicle and pedestrians). All available maps in Carla, of any road or side walk traffic, are free. That's why all vehicle or pedestrian traffic needs to be spawned on the map. The initial position of actors such as pedestrians is defined by spawn points and xyz offset to this point. Spawn points are the predefined positions located on the virtual map to simplify the process of actors relocation. Spawn points are categorized in such a way to determine if particular point is for pedestrian (sidewalk or other spot available for people) or for vehicles on the roads. There are hundreds of different spawn points on each map. By randomizing the spawn point number for each actor per run, unique initial scenarios are generated automatically. Not only the place and pose of the pedestrian is important to diverse the scenarios. People body can be taller or shorter, thinner or coarser. Those variables are also simulated by Carla by the range of multiple human body versions. Each version of human has its animation and movement defined. Also children smaller bodies are simulated. Carla delivers around 40 different body shapes with their animations. Figure 42 depicts some example of those pedestrians bodies.

Figure 42: Example of different human bodies available in Carla simulation used for LiDAR data generation.

To generate different scenarios, which include pedestrians, all spawn actors have to be moved. Individual control of each pedestrian or vehicle, would be far from the optimal solution, so simulator developers created the 'auto pilots' for them. For each walker actor, the automated controller can be assigned to move them around the map. Each map has predefined roots, where pedestrians can walk. Walker controller can be set to walk to the certain destination point or it might be set

to the ‘auto’ mode. This mode makes the actor walk around the map in the endless loop, between different spawn points.

The last part of the preparation for data recording is to spawn ego vehicle with the necessary sensors. As it was already mentioned, each sensor has to be attached to an actor. This actor movement makes also the sensor move around the map. For this research, the Tesla model 3 was selected as an ego vehicle, which carries the LiDAR sensor. Sensor parameters were set similarly to HDL-64E LiDAR, described in table 4. Spawned pedestrians with auto pilots set were generating random traffic on the sidewalks and the ego vehicle equipped with LiDAR sensor could scan those scenarios to generate required point cloud data. Data recordings were performed on all of 7 maps, available by default in Carla.

6.3.1 Automated labeling of simulated objects

Generated point cloud is only the one part of data needed to perform supervised learning process. Each pedestrian, on every single frame, needs to be precisely labeled. Labeling is the most important information for the neural network learning algorithms. Model weights update is based on the prediction and ground truth provided for each data frame. Precise bounding boxes for objects is the key, to perform the proper learning process. In the standard approach with the real sensor data, each object of interest, for example pedestrian, has to be labeled manually by human. Humans are able to detect and mark all the required objects, based on their life experience. This type of labeling process is expensive and time consuming. To overcame this issues, synthetic data has to be automatically labeled [65, 66]. Since all objects in the simulation are controlled and located in known places, it is possible to calculate the bounding boxes for each of them at any time. The algorithm which automatically calculates the labels for objects in simulation was developed for the purpose of this research.

The first step to generate the labels for range images of LiDAR point cloud is to get the size of the objects in simulation. This information is provided by simulation for each model of the object. This property of object is called extent and includes values for each dimension: x, y, z, in other words width, height and depth. Next information needed for computation is the position of the sensor and the position of the pedestrian. Each object position is defined by two 3D vectors. First one is the location vector and the second one is the rotation vector. The computation process is split into a few smaller steps. Bounding box size: width, height and depth is defined with respect to the center of the pedestrian. First step is to build 8 point array, which describes the bounding box corners. Position of the actor bounding box is relative to the Carla world, so in the next step the array defining the bounding box is transformed the coordinates in the Carla world. The goal was to get the bounding box coordinates relative to the sensor position, so the next transformation: from the world to sensor position coordinates was required. It was performed on the basis of the information concerning the sensor location. Last step of computation was to transform all 8 bounding box corner points to the spherical coordinates, as it was done in case of the range image conversion.

To test the developed algorithm, it was used on the camera sensor. To validate the positions of the bounding boxes of detected objects, the output of the algorithm was drawn as 2D representation of 3D bounding box on the camera RGB image. The examples of three calculated bounding boxes of parked cars are depicted in figure 43.

Figure 43: Example of automatically generated bounding box projection onto the camera image sensor data. Green lines depict the sized and position of each vehicle 3D label

In figure 43, it is visible that automatic labeling algorithm is working correctly. Each vehicle has a precises 3D bounding box drawn around it's model. This method can be used as the source of LiDAR range image ground truth. Description of the algorithm work was provided for only one standing actor and standing sensor. For the purposes of data gathering, all object are moving around the map and there are numerous pedestrians. Inputs for the algorithm are changing for each frame of the simulation. Since the LiDAR sensor rotation speed is set to 10Hz, simulation frame rate is also 10. Algorithm must efficiently calculate the transformation of each pedestrian into the sensor position, ten times per second. Camera example 43 includes 2D projection of 3D bounding box. Range image for the LiDAR data is also 2D, but for the supervised learning process only 2D label is needed. To see how the label fits the pedestrian on the range image, and example in figure 44 is shown. Range image example is the same scenario as one presented in figure 39.

Figure 44: Example of automatically generated bounding box projection onto the LiDAR range image. Black box depicts the size and position of the pedestrian on the image.

Before the generated labels can be used as the input data for the model training, they have to be reshaped. YOLOv4 allows to change the input image size to fit the model into the given detection task. To let the learning algorithm work on the ground truth labels of different size, they have to be defined with respect to the input image size. Figure 45 shows how the bounding boxes are defined for the YOLOv4 model.

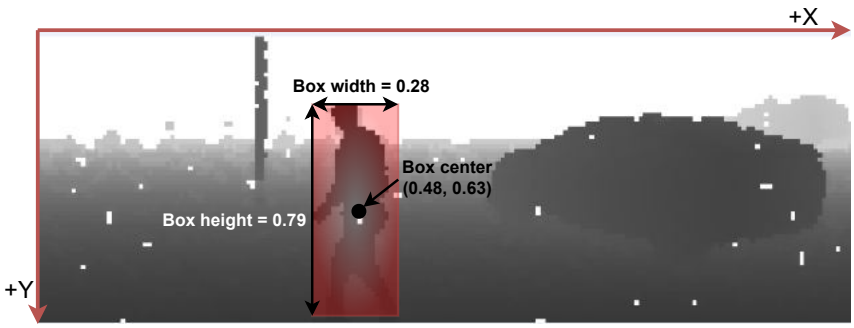

Figure 45: Image description of how YOLOv4 input bounding boxes shall be described.

Each input frame should be delivered with the text file, where the bounding boxes of objects are defined: one line of text for one bounding box. Line should start from the number of class for particular object. In case of our detector there is only one class: 'pedestrian'. Therefore the first value is zero. Further values are the bounding box parameters 45. Box center point coordinates followed by width and height of the box. It is important to remember that the text file associated with the image has to be named exactly the same as image. Process of reshaping automatically generated labels from simulation into YOLOv4 format is included in the process of data generation.

6.3.2 Examples of synthetically generated LiDAR data

The first training experiments were performed to determine if the synthetic data with automated labels is a valuable input for deep neural network model. Over one thousand of range images with labels for pedestrians was generated for this first experiment. Scenarios include diverse road scenario where people are crossing road, walking on the side of the road and other normally encountered road situations. First example photo (figure 46) depicts four different scenarios generated for this research. For better understanding of scenario, it's shown as front camera image.

Figure 46: Four examples of road scenarios generated in Carla with walking pedestrians. Scenarios include pedestrians crossing the road, walking on the sidewalks on the left or right side of the sensor. Parking scenario, where the pedestrians walk between parked cars, were also prepared and recorded.

Next figure 47 depicts several range images generated out of the simulated scenarios. All pedestrians visible on the examples are automatically labeled. Labels are drawn as black rectangles.

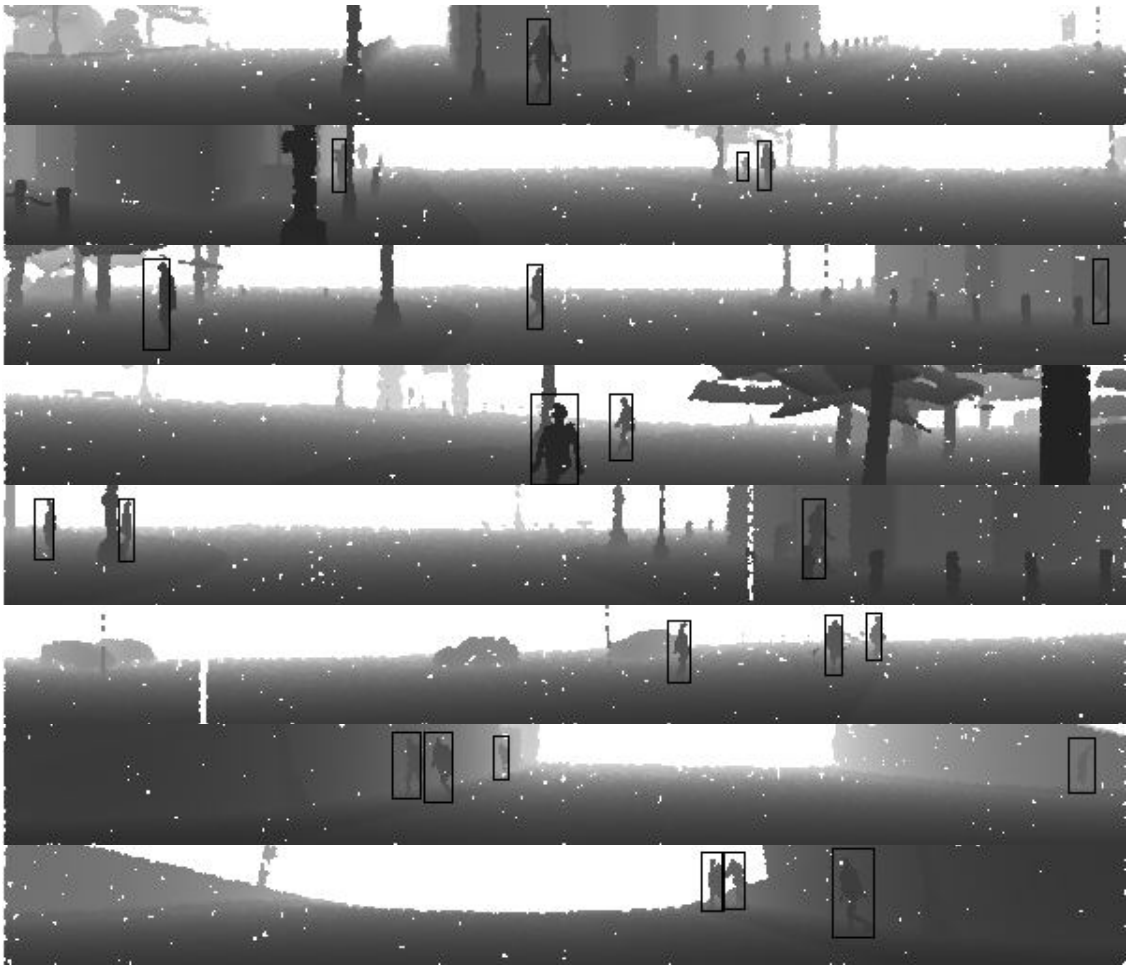

Figure 47: Several examples of range images generated out of the simulation of LiDAR sensor with pedestrians bounding boxes drawn.

On the example range images, there are technical issues of data gathering process visible. In some pictures, white vertical lines appear. Those spots are related to the fact that the simulation frame generation is not exactly synchronized with the sensor rotational speed. It means that if the simulation server renders one of the frames a bit too fast, it will result in loosing part of point cloud data. In such a case, the ray-casting method would not be able to catch the full field of the sensor view and, if this part is in front of the sensor view, it appears as white missing part of data. The second issue is related to the position of the pedestrian bounding box. Some are a bit mismatching the real position of person on the image. The precise positioning of labels is crucial in terms of machine learning process. To reach the high detector performance, the labels should be as precise as possible. The root cause of this issue has to be found and fixed, if it is possible. The main finding was to realize that the level of mismatching was increasing while the sensor rapidly changed its position relatively to the labeled object. Figure 48 shows more precisely the examples of mismatched labels on pedestrians.

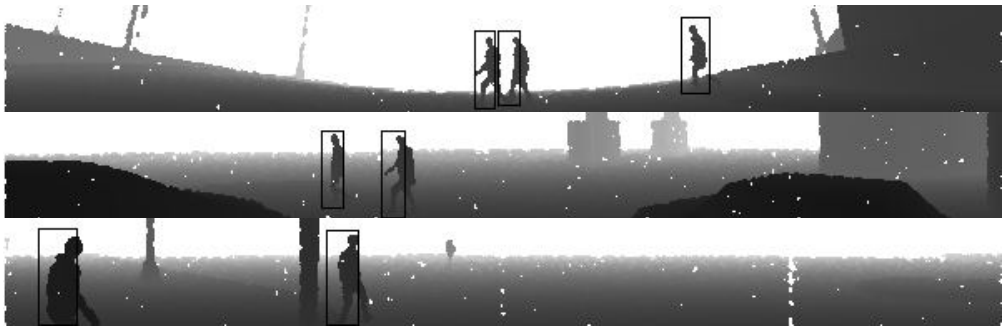

Figure 48: Examples of mismatched labels on pedestrians, caused by sensor movement relative to the objects.

After investigation, the root cause of the inaccurate labeling was found. The input positions of the sensor and pedestrian were a bit delayed in comparison to the ray casting of LiDAR sensor. The process of LiDAR data generation and object position updates are separated in simulation. Ones the algorithm was informed about the object position, the LiDAR was scanning it in a bit different location. Missing synchronization between simulation process made it impossible to precisely label sensor data. The only solution was to record objects without movement of the sensor. The output was still realistic due to pedestrian movement, but the process of data gathering was extend in time. Results of conduct experiments show the impact of those imperfections on the detector output. It is worth to mention that this type of problems was found also for other computer simulations. In publication [8] researchers describe exactly the same issue with synchronization of object position and LiDAR data. Their solution for the problem was to stop the simulation each time the data gathering process was run. This solution can significantly improve the process of generation of synthetic data. In this research, the new solution for automatic labeling will be introduced, which does not require to stop simulation on every single frame.

6.4 First pedestrian detector training, performed fully on synthetically generated data

In the first pedestrian detector training only synthetically generated data were used. For this experiment, over 1300 range images were generated and processed. Generated data were split into two datasets, training and validation. The split ratio was set to 80/20%. Darknet AI framework was used to train detector. Fine tuning method [67] with pre-trained weights *yolov4.conv.137* [68] were used for pedestrian detector. Weights file is the output of the training process. This file describes the weight values of each neuron in the network architecture and connection between the neurons Fine tuning method freezes the pre-trained layers of the convolutional neural network, which were trained on the large dataset. This method allows to remember the feature extracting layers to rapidly fit the last detection layers to particular objects. In the considered case, detection should be performed for the pedestrians visible on the range image. Method is also known as transfer learning. Method transfers learned abilities of network to new object detection tasks. A few training parameters must be set before the process can be started. Most important is to fit the size of images to the YOLOv4 input size. Network architecture allows to change the size of input but within defined rules. Image input size must be the multiplication of 32 pixels. Height of the

generated range images is 64, which is exactly double of 32. The width is 600, which is not a multiple of 32 value. The closes possible width is 608. Most of other training parameters defined in the configuration file for the Darknet framework are default. Table 5 describes most important parameters set for this training process.

Table 5: Darknet training configuration for a custom LiDAR dataset.

Parameter	**Set value**	**Description**
Input width, height	608x64	Image size used as input for the network
Learning rate	0,001	Initial learning rate for training algorithm
Burn in	1000	Initial weights burn in value
Max batches	6000	Amount of batches processed in the training
Steps	4800/5400	Two step thresholds on which learning rate will be multiplied by a scale factor
Scale	0,10/0,10	Scale factor for training rate multiplication 'Steps'

While the training process is run, after *burn in* batches, the framework saves the model weights. Those weights can be used to compare best performing model with this from the earlier stages of the training. All necessary parameters and input information are set, so training process can be started.

6.4.1 First pedestrian detector training results

Training process in Darknet framework is depicted as a chart. On this chart the number of processed batches versus loss values is plotted. Additionally, after around 600 batches, the algorithm calculates the percentage mAP values of the actual model weights. Framework overwrites the weight files called 'best' each time model performance indicated by mAP value is better than the last best one. Figure 49 depicts the training graph out of the experiment training done on synthetically generated data. Validation dataset is also taken from the same synthetic dataset.

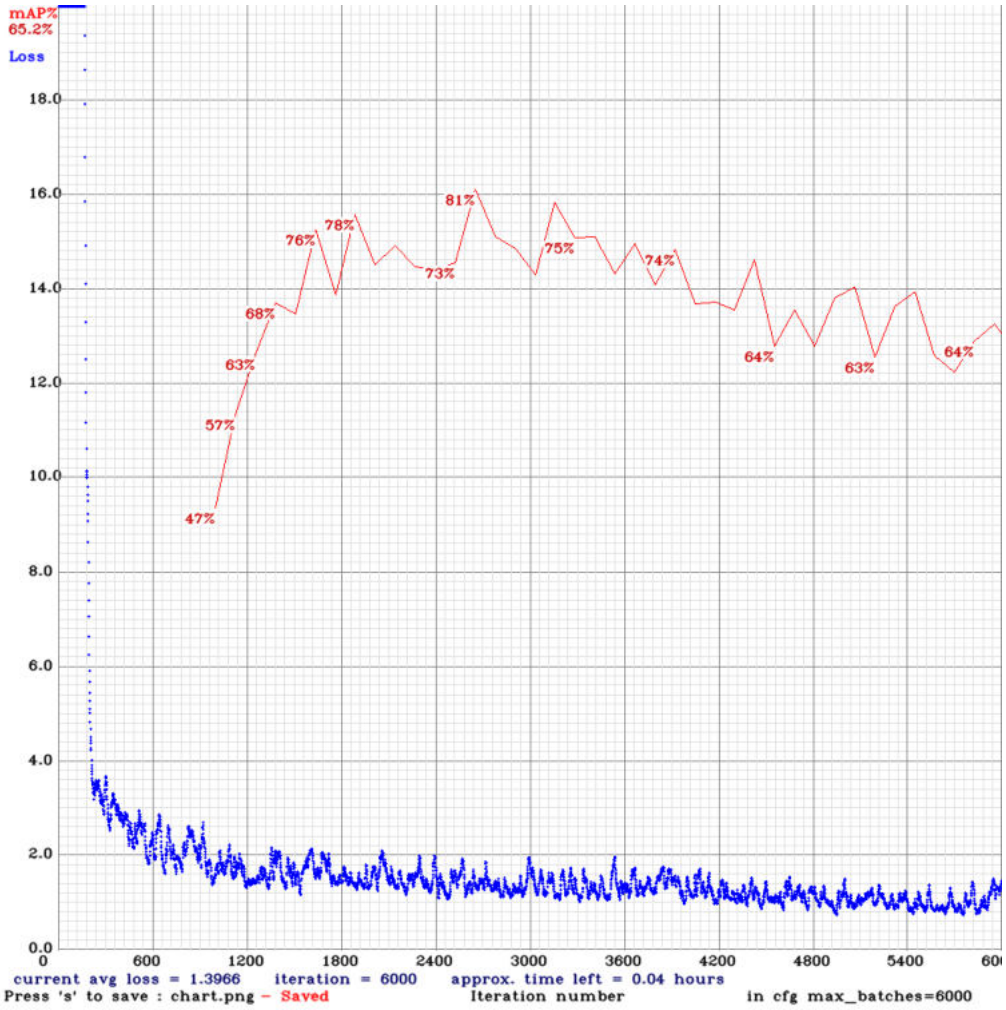

Figure 49: Training progress chart, for Yolov4 pedestrian detector. Training validation calculated on synthetic data.

For the first calculated mAP value the model reached over 47%, which is a really high level, considering that this was just a beginning of the training. Such a high level, could be reached thanks to the fine-tuning method used. Pre-trained weights allow the model to really fast find the features, which determine the position of pedestrian on the image. Proper preparation of the input data resulted in over 81% of mAP. mAP value for training is calculated for IoU threshold set to 0.5. After around half of the training batches, model starts to overfit [69]. This phenomenon appears while the model starts to learn exactly, how the training data looks like and this leads to worse performance on not known validation data. Overfitting starts when the loss value out of training is dropping while AP also starts dropping. Lowered loss does not contribute to the model performance indicated by mAP. On the training output plot 49, the framework indicates also the actual iteration and approximate training time. Actual value of the average loss is also indicated on the bottom of the plot.

To be able to understand the performance of the trained detector, charts with the information concerning precision, recall and F1-score were prepared 50. All statistics are the function of the confidence threshold.

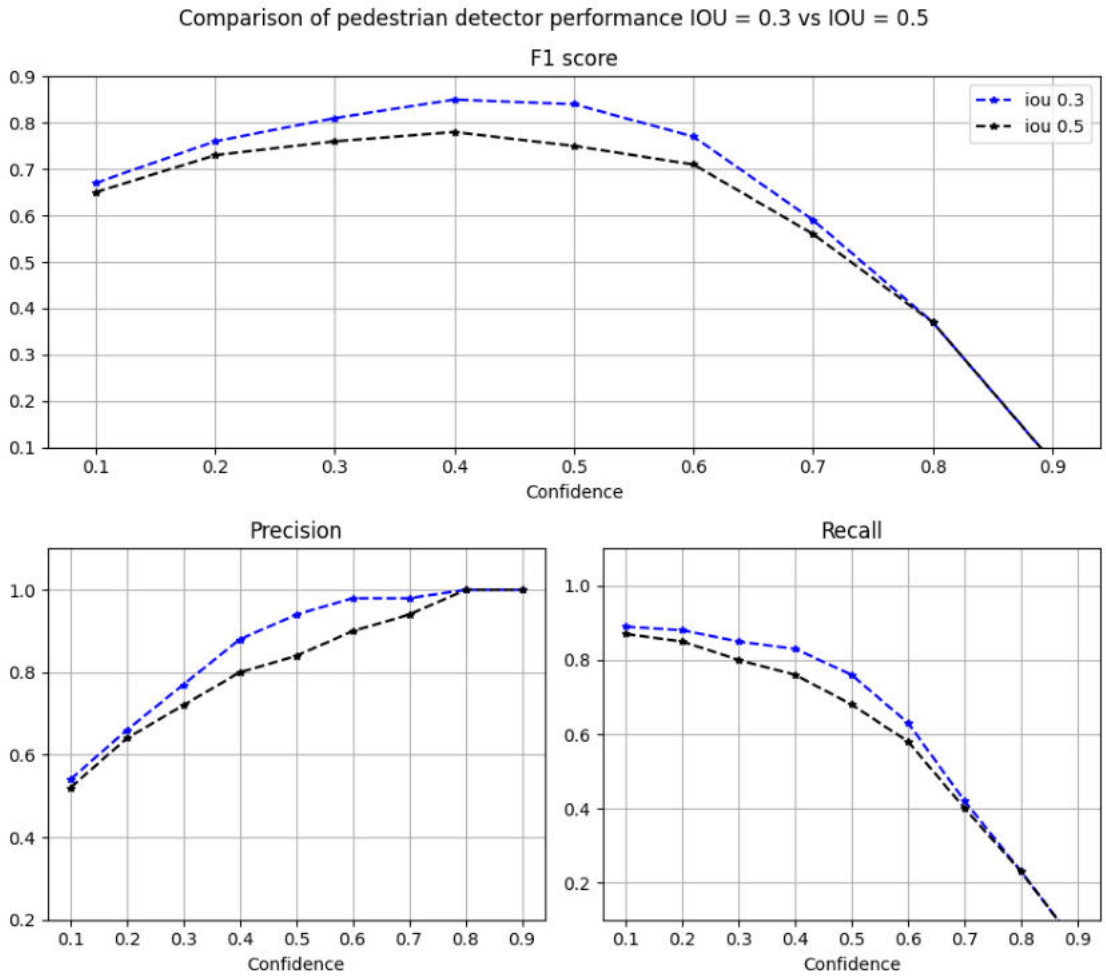

Figure 50: Pedestrian detector performance charts for precision, recall and F1-score. To compare the results obtained for IoU = 0.5 and IoU = 0.3 both statistics were plotted.

Figure 50 compares the performance of the detector for two different values of IoU, to show that model has a significantly better performance for a bit lower value of IoU. Precision curve shows almost 10% higher value for the confidence threshold 0.5, than in case of the standard IoU = 0.5. Overall, F1-score also indicates much higher performance of the detector for the lower value of IoU. The next plot (figure 51), depicts the precision-recall curve with mAP indication for both values of IoU.

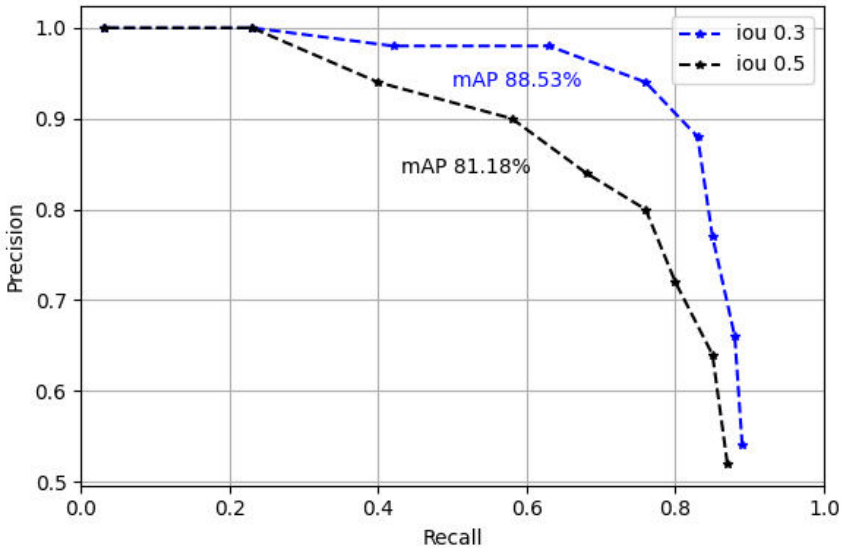

Figure 51: First pedestrian detector precision-recall curve with mAP values for two IoU = 0,3 and 0,5.

Model evaluated for IoU = 0.3 indicates over 7% higher average precision value than for IoU = 0.5. Significantly better performance for low values of IoU means that generally model detects pedestrians correctly, but the positions of the bounding boxes are not precise in comparison to the ground truth. The root cause of this situation is that the automatic labeling of pedestrians on the training and validation datasets is not precise 48.

Trained model indicates quite high performance, so let's see how it works on the example range images. The first two simple scenarios with one or two pedestrians which are correctly detected are presented in figure 52. All shown examples are taken from the validation dataset not known to the model

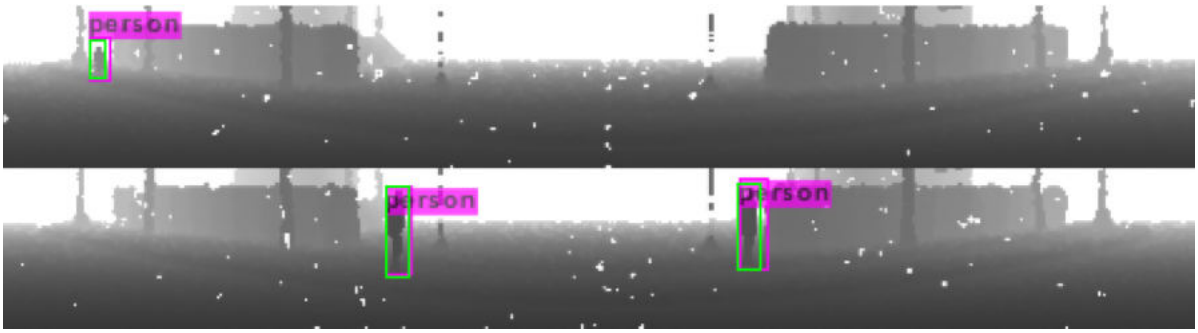

Figure 52: Detector examples of correct detection. Image depicts the prediction purple labels and ground truth green labels

Examples in figure 52 depict comparison of the ground truth green labels and prediction purple labels. I is visible that the labels are mismatched a bit, but in this particular examples the error is not too big. Next examples, in figure 53, depict the situation where detector does not predict position of all pedestrians on the scene.

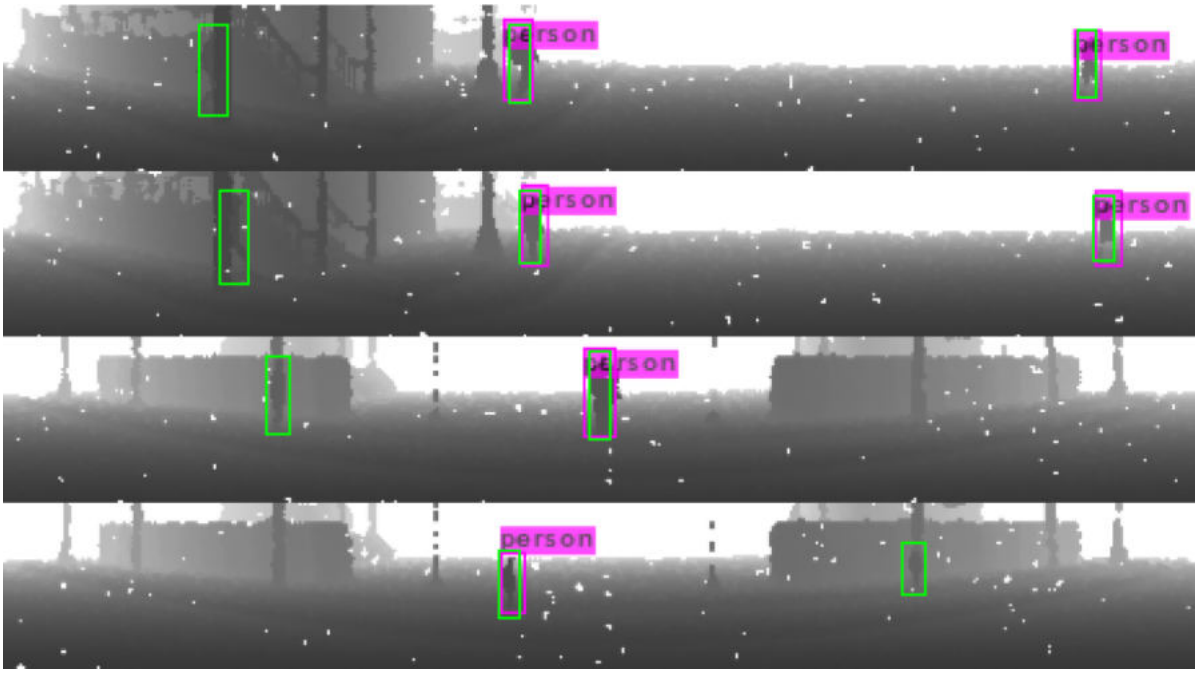

Figure 53: Detector examples with incorrect pedestrian detection. Image depicts the prediction purple labels and ground truth green labels

In figure 53 missing detections result mostly from the fact that the pedestrian is obscured by some other objects, like a street pole. All other clearly visible pedestrians are correctly detected on the scene. Again, as in the first example, prediction labels are mismatched with respect to the green ground truth labels. Last examples in figure 54 depict the situation, where model predicts nonexistent pedestrians or it doubles the number of them at the same position.

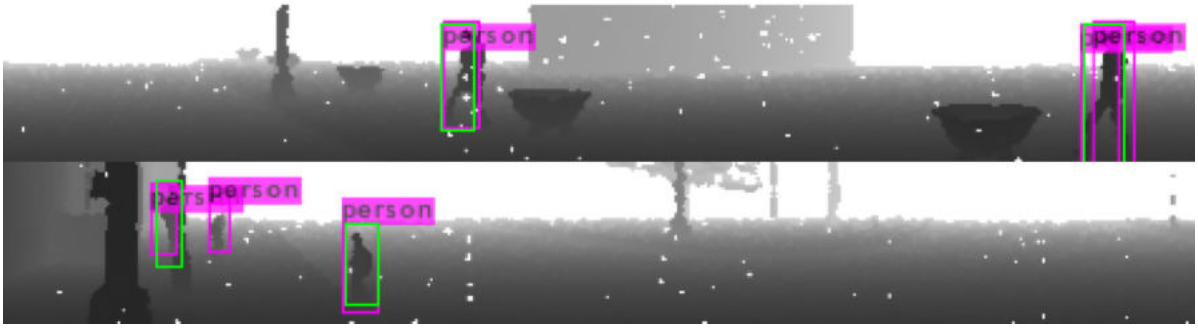

Figure 54: Detector examples with incorrect pedestrian detection. Image depicts the prediction purple labels and ground truth green labels

Examples from figure 54 depict so called false positives. On one of the examples a street pole was detected as pedestrian. On the second example one pedestrian on the sidewalk was detected as two.

Feasibility study on the synthetically generated LiDAR data lead to the high performance detector of pedestrians out of range images. Considering imprecise labels for training, detector

reached 88.53% of mAP value for IoU = 0.3. Detector training experiment shows, that synthetically generated data, out of Carla simulation, is good enough to properly train the YOLOv4 deep neural network. For the further experiments, an improvement of automatic labeling method is necessary. Comparing standardized statistics of neural network trained to detect and classify objects, the model should not be evaluated for such a low value of IoU. The second improvement needed in the process, is to synchronize sensor with simulation frames to fix the issue of data loose (figure 47). A kind of data buffering can help to correctly generate full LiDAR field of view data. To determine fully the usability of synthetic data it has to be evaluated on the real LiDAR sensor data.

6.5 Preparation of real Waymo dataset for models validation

The idea behind synthetically data generation is to improve the process of supervised learning process. Improvement can be done by speeding up the data generation process or by improvement on the model performance thanks to the new important scenarios, which are hard to be recorded on the real roads. Simulation can also reduce the work load required to achieve the goal. To assess whether the synthetic data can improve the process of supervised learning it has to be validated on the real sensor data. Otherwise, high performance detector, like the one described in the previous chapter, might not work correctly in the real vehicle application. Real LiDAR data is needed to build a validation and comparison dataset, which might be used for pedestrian detection out of road scenarios. There are few open source datasets which includes LiDAR sensor data gathered on roads. Most popular one is Kitti [4], which include around 15 k of LiDAR frames gathered on road round the mid-size city of Karlsruhe. There are also datasets like NuScenes [5], with even bigger LiDAR datasets. The newest and biggest dataset is the one recorded by Waymo company. Waymo dataset includes over 200 k LiDAR frames and almost 10 M of labels. Additionally, the LiDAR point cloud from Waymo dataset is also available as a range image projection [70] – the data format is the same as the one developed out of Carla simulation. Waymo dataset includes not only one sensor data. There are data out of 5 cameras and 5 LiDAR sensors recorded at the same time. Dataset is divided into thousands of .tfrecord files. Each file includes compressed data out of around 200 frames of each sensor. Highest resolution top LiDAR, which has 64 channels, is the only LiDAR sensor from the Waymo dataset, which were used in this thesis for pedestrian detector experiments. Sensor is visible in figure 1 on the vehicle roof. To be able to extract the necessary data, the TensorFlow Python package was used. There is also a dedicated package for the Waymo dataset, which includes examples and functions to work on the compressed data. For this research *waymo-open-dataset-tf-2-6-0* package version 1.4.1 was used [71]. Figure 56 depicts the example range image generated by Waymo dataset, based on the top LiDAR point cloud. To understand the scenario scanned by the sensor, images out of 4 cameras are depicted in figure 56. Both figures were prepared for the same recording time.

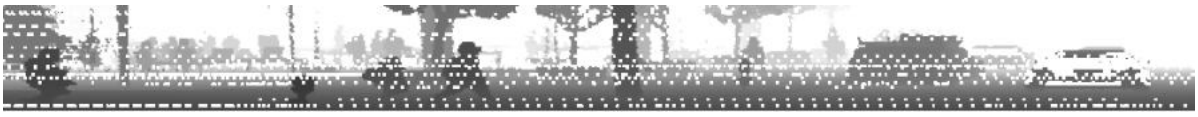

Figure 55: Example of raw range image generated out of the Waymo dataset top LiDAR sensor. Image includes only a part of the sensor field of view. Waymo range images are deliver for 360 degree view around the vehicle. The whole image has over 2600 pixel width.

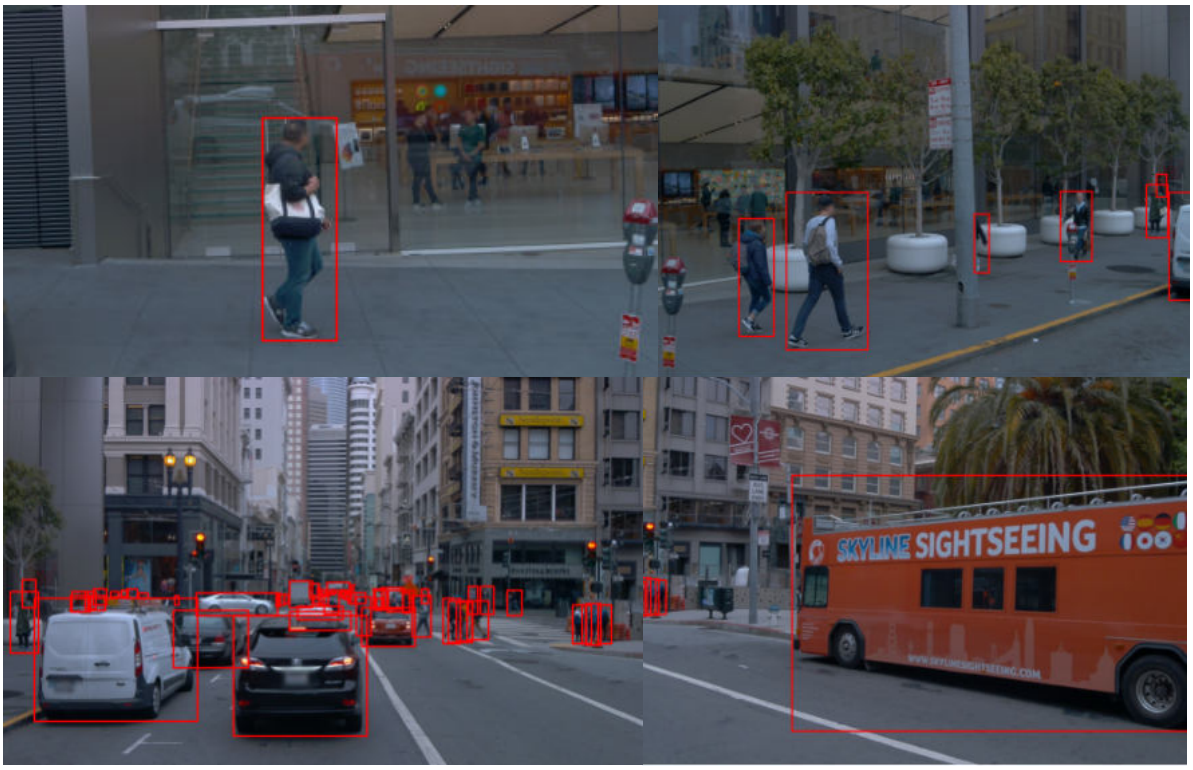

Figure 56: Camera sensors view out of the Waymo dataset from the same time instant as on the range image 56. Camera images are delivered with 2D bounding boxes for all the objects like pedestrians, vehicles or traffic signs.

It is needed to mention about Waymo sensor parameters. Most important parameter for a range image is the vertical field of view (VFOV) of the sensor. The top LiDAR of Waymo vehicle has 20 degree of VFOV, where mostly it aims down from -17,6 to +2.4 degree. In comparison to HDL-64, it has 5 degree smaller VFOV, see Table 4. Regarding the rotational speed and resolution, the Waymo LiDAR has a similar performance as Velodyne HDL-64E. Table 6 shows all the parameters of the top Waymo vehicle LiDAR.

Table 6: Waymo top LiDAR parameters.

Parameter	**Waymo Top LiDAR**
Number of Channels	64
Measurement range [m]	75 (restricted)
Vertical FOV [◦]	+2.4/-17.6
Horizontal FOV [◦]	360
Vertical angular resolution [◦]	∼0.136
Horizontal angular resolution [◦]	0.3125

Vertical angular resolution is not provided by the Waymo company, its value is calculated out the horizontal FOV and the number of pixels defined in the range image. Other parameters of sensor are not known, but those which are crucial for the research are known.

6.5.1 Processing of LiDAR data provided by Waymo dataset

Raw range image generated from the considered dataset is really wide. Full 360 degree view of LiDAR projects to range image size of 2650×64. Similarly to the first raw range images generated out of Carla sensor, real range image is also grainy 39. Real sensor is not able to receive perfectly all the casted rays, so the white points appear on the image. On the right side of the example image, a car in front of the sensor is visible. There are lot of missing points from this spot. Glass rear window of the car does not reflect LiDAR points. Real LiDAR sensor dataset delivers 3D bounding boxes for the point cloud, but there are no valid 2D projections of this labels onto the range image.

Range images delivered by Waymo dataset require similar processing as the synthetic data from Carla, Figs. 39, 40,41. To be able to manipulate the data shape, all points from the cloud were processed in the same way as the data points from Carla simulation. As the result, files with points defined in the spherical coordinate system were generated. The first post processing step is to truncate the field of view to front 180 degrees. Example image, after the first step of processing, is depicted in figure 57.

Figure 57: Waymo dataset example range image truncated to front 180 degree view.

The second step of the post processing is to double the horizontal resolution. Two near 64 pixel columns are merged into one column. This reduces the width of the image by two. The output image has 662×64 pixel size. Same example as in figure 57 is depicted in figure 58 but after the change in the horizontal resolution.

Figure 58: Example range image from Waymo dataset after two stages of post processing.

The last step of the range image processing is to set a median blur filer with 3 pixels of radius. The process is the same as for the synthetic Carla data. This step removes all the white missing points out of the image. Figure 59 depicts the result of all processing changes done on range image.

Figure 59: Example range image from Waymo dataset after all post processing changes.

Application of the post processing methods to the input Waymo data results in the smooth and sharp images, where pedestrians and other objects are clearly visible. For all experiments, over 6 k range images were generated out of the Waymo dataset. Out of all generated data, around 600

frames were choose randomly as the validation dataset. All further experimentally trained detectors were tested on this validation dataset, generated from the real sensor data. Figure 60 depicts six different example range images generated for the real sensor training data.

Figure 60: Six example range images choose from generated dataset.

Dataset includes variety of road scenarios with pedestrians. At some scenes, there are dozens of pedestrians on the crowded road crossing. Scenes where no pedestrians are visible are also included in the dataset. The poses of pedestrians in the dataset are diversified, people are walking, standing or sitting. Some frames are gathered during the rainy days [72]. This situation introduces an important modification of the pedestrians shapes. The change is related to the additional items carried by people, such as umbrellas or hoods. Example of such image is depicted in figure 61.

Figure 61: Scenario recorded at rainy day, where pedestrians carries the umbrellas.

On the LiDAR range image this type of pedestrians which are carrying umbrellas looks totally different than usually. This type of diversity of object shapes, makes it more challenging to train

the detector. Moreover, right now Carla simulation does not allow to create similar scenes where pedestrians are carrying items in hands or on the back.

6.5.2 Preparation of objects labels for range image projection

Range image is one part of the input data for training. All pedestrians have to be correctly labeled on every single frame out of 6k of frames. Waymo dataset is delivered with object bounding boxes but not for the range image projection. For cameras there are 2D labels for each sensor and for LiDAR 3D labels are defined. Labels are created for vehicles, pedestrians, cyclists and signs. Each label has its unique ID, which might be used for object tracking tasks. Figure 62 depicts an example 3D point cloud image with bounding boxes drawn on objects.

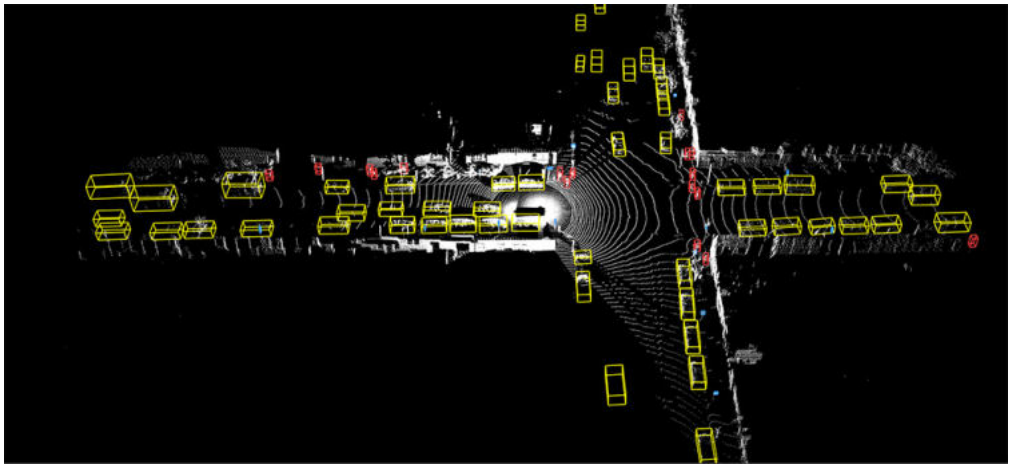

Figure 62: 3D Waymo point cloud example with bounding boxes drawn on objects [73].

To map 3D LiDAR bounding boxes on the range image the projection has to be applied. Output of the algorithm should include position and size of label defined in pixels. Developed algorithm is based on the input of 3D labels defined in Cartesian coordinates and special range image interpretation defined as 2D array which includes x, y z points. This interpretation can be shown as 2D matrix, where each element is a 3D point defined in Cartesian coordinate system. Table 7 depicts how this special interpretation of range image is defined.

Table 7: Definition of range image interpretation which include 3D Cartesian points.

	$Column_1$	$Column_2$	$Column_i$
Row_1	$p_{1,1}(x,y,z)$	$p_{1,2}(x,y,z)$	$p_{1,i}(x,y,z)$
Row_2	$p_{2,1}(x,y,z)$	$p_{2,2}(x,y,z)$	$p_{2,i}(x,y,z)$
Row_i	$p_{i,1}(x,y,z)$	$p_{i,2}(x,y,z)$	$p_{i,i}(x,y,z)$

These type of point cloud assignment, as defined in table 7, allows to determine which point of data creates which pixel on the range image. Algorithm iterates thru columns and rows of special range image. Second input is a particular label box. For each point in Row_i and $Column_i$ the algorithms calculates, if exact point lays inside or outside of the bounding box. Each point

which is inside has a corresponding Row_i and $Column_i$ in range image and this information is accumulated. After the iteration, output of this loop is a set of pixel coordinates of points which lie inside the bounding box. The last step is to calculate the maximum and minimum pixel coordinates for this label. That's how the algorithm defines pixel coordinates of the edges of the bounding box for each object. Algorithm loop is repeated for all the labels which exist in the input frame.

There are situations when one object is covered by the second one. This kind of situation is common in crowded places like pedestrian crossing. Two labels on the same pedestrian might confuse the training algorithm of neural network. To overcome this problem, the filtering process on each frame label is performed. For all pedestrian bounding boxes defined for the particular frame, the IoU is calculated between all the boxes. This allows to determine if such a situation exists. Experimental value of IoU = 0.7 was selected to filter out the bounding boxes which covers each other. Average distance to the point inside the bounding box was calculated to determine which box should be filtered out. Bounding box which was further was chosen to be removed from the data. The second type of label filtering was the one set on really small bounding boxes. The rule, which removes the labels with too low amount of points inside, was developed. The removal was determined by the distance to the object. Distance to the pedestrian was split into three categories: close, mid and far. Each category has a specified value of point which has to be included inside the bounding box, to let this label exist in the dataset. The last step to prepare the correct definition of the object label is to reshape it to the YOLOv4 format, see figure 45. Reshaping is exactly the same as for the synthetic dataset. The effect of 3D bounding box projection onto range image is depicted in figure 63.

Figure 63: Same range image example of Waymo dataset as on the figure 60, but with black labels drawn on each pedestrian.

One of the most difficult challenges for the detector is to distinguish pedestrian from cyclist. On the fourth example image, in the middle of the scene, there is a cyclist visible. As this is not a pedestrian, it is not labeled. The issue is that regarding the shape it is obviously person but sitting on the bike. That is why a lot of cyclists is going to be detected by prediction algorithm as pedestrian. More details about the training results on the real dataset will be descried in the further chapters of this thesis.

6.6 Synthetic data fitting to the validation dataset

6.6.1 Validation of first synthetic data detector on real LiDAR data

Real sensor LiDAR dataset is prepared. For pedestrian detector validation 600 range image frames were choose randomly. Right now it is possible to determine how good is the pedestrian detector on the real LiDAR data. Detector validated on synthetic data indicated mAP of 81.18% for IoU = 0.5. Figure 64 plots the precision, recall and F1-score of the synthetic detector validation on the real Waymo data.

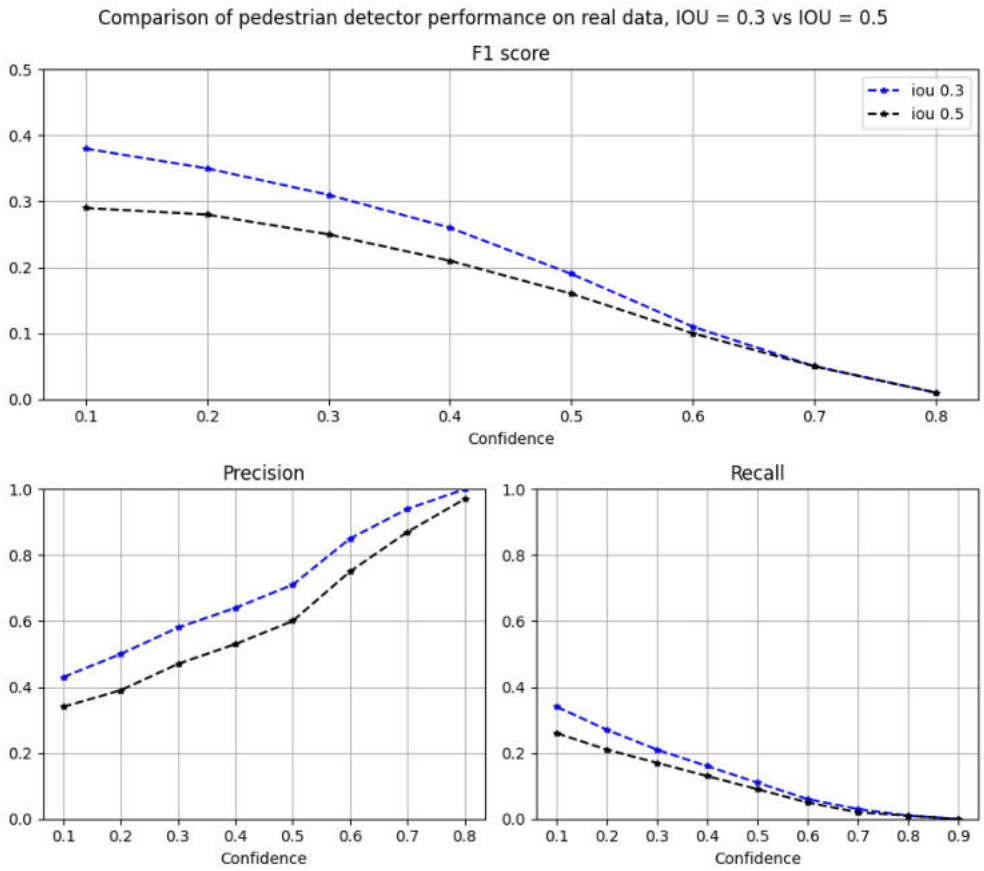

Figure 64: Pedestrian detector performance charts for precision, recall and F1-score, validated on real data. To compare the results obtained for IoU = 0.5 and IoU = 0.3 both statistics were plotted.

Detector validated on the real data indicates terribly low results. The highest F1-score is around 0.4, which mean less than half of the value in comparison to the one calculated on the synthetic

data. To fulfill the statistics of this detector on the real data, figure 65 plots precision-recall curve for two values of IoU.

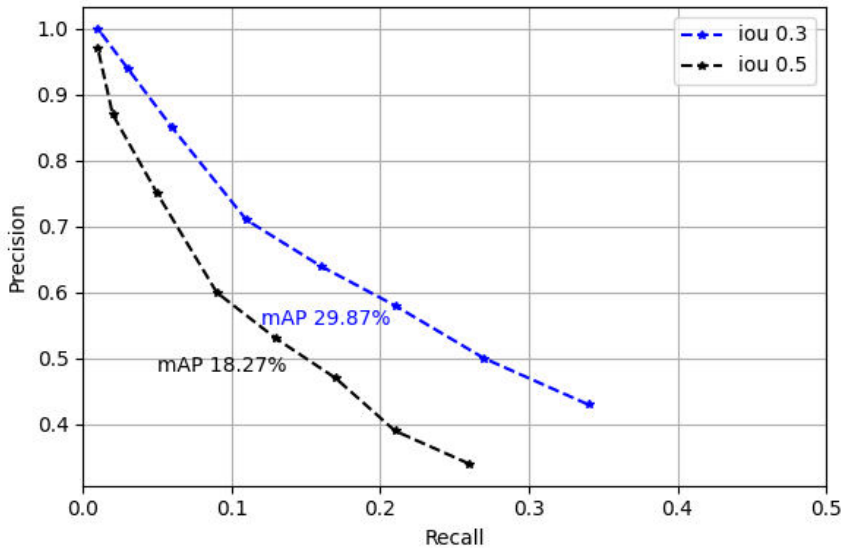

Figure 65: Pedestrian detector precision-recall curve with mAP values for two IoU = 0,3 and 0,5. Detector validated on real data

Detector indicates poor performance on the real LiDAR data. This is an experimentally achieved result, which shows phenomenon called *Synthetic to real gap*. Similarly bad results were described by researchers in publication [38]. Without fine tuning performed on real data, detector is totally useless in the real applications.

Those poor results are not surprising. Synthetic data were generated based on different sensor parameters and located in other place of vehicle. Different field of view and location of the sensor, created different perspective of road scenarios and shapes of pedestrians. Character of synthetic versus real data is so different that detector does not recognize pedestrians. To see the differences between the real and synthetic data, the figure 66 is provided.

Figure 66: Comparison of first synthetic data to real sensor data.

Real sensor data, after application of all post processing methods, does not include any grainy spots, which exist in the synthetic data. Next problem with synthetic data is the precision of automatic labeling. This issue, described in the chapter 6.3.2, may have a significant impact on the

detector performance. A lot of work was put to deal with all mentioned problems and generate synthetic data better fitted to the real Waymo dataset.

6.6.2 Simulated LiDAR sensor, model fitting

LiDAR model simulated in Carla environment is parametric. The first step to fix the quality of the synthetic data in comparison to the real one, is to modify the parameters of the LiDAR model. Vertical and horizontal field of view, as well as resolution, were set the same as for the Waymo top LiDAR, see table 6. The results of those changes are depicted in figure 67.

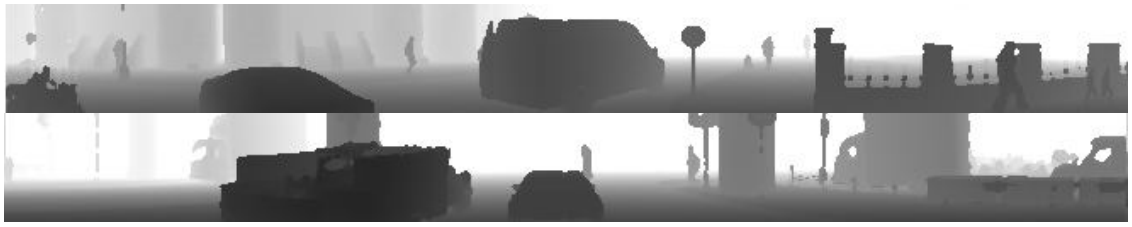

Figure 67: Comparison of fixed synthetic data to real sensor data.

Proper setup of the LiDAR sensor in simulation created range images close to the real data. It is hard to determine which image comes from the real sensor and which from simulation (figure 67). Introduction of changes in the simulated sensor setup was the first step to improve the quality of the synthetic data. The second important change, regarding the automated labeling process, is much more complex. Any movement of sensor against labeled pedestrian makes it mislabeled in the output data. To overcome this problem, a new innovative solution was developed.

6.6.3 Semantic segmentation LiDAR solution for automated labeling

The main idea behind a new method of automatic labeling is to overcome the problem of time differences between the real time of LiDAR scanning and the simulated world time. Separated processes created problem with synchronization between object position in simulation world and in the sensor data. The novel method was developed to overcome this problem and deliver precise automatic labels for all types of objects in the sensor view range. The crucial change in Carla simulation gave the possibility to develop such a solution. This change was to create the semantic segmentation LiDAR model (SS-LiDAR). Since Carla 0.9.11 version, the SS-LiDAR is available to be used. This sensor works exactly the same as standard LiDAR, except of the amount of information provided. Each SS-LiDAR point includes standard x, y, z position coordinates and, additionally, the tag and ID of the object which was hit by this point. Segmentation tag is an information about the type of object or surface. Each object of structure in simulation has its defined tag assigned. This tag gives precises information about type of object hit by the ray-casting process. The second additional information is the ID. It is unique identifier of the object in simulation. Each vehicle, pedestrian, sign or other object has its unique ID number. New method of automatic labeling is based on the point tag and ID information of hit object. New algorithm uses only point cloud data generated by the SS-LiDAR to generate range image and label the objects. Auto labeling algorithm is divided into a few steps. First is to generate a standard range image out of the input point cloud. The next step is to generate empty images where only points which tags

are equal to the tag of pedestrian is marked with red color. This means that the algorithm creates separate images of each pedestrian visible on the scene. The next step is to calculate the size of the pedestrian, based on the edges of the red colored pixels. This edges create bounding boxes for each object. Accumulated boxes are saved as ground truth for the range image. The source of the label and range image is the same, so the effect of this method application is almost perfect. Graph of the work of the new algorithm is depicted in figure 68.

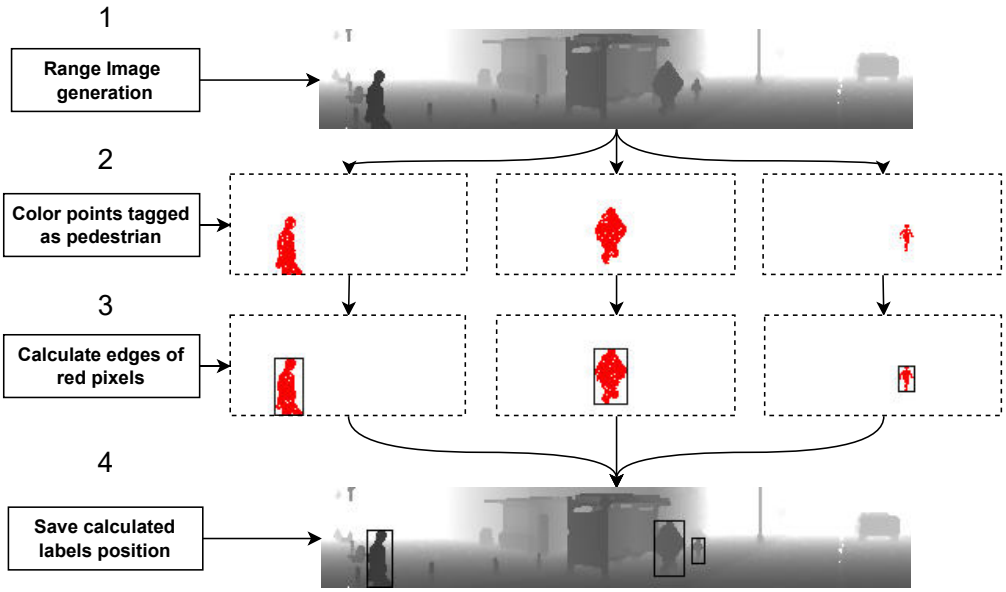

Figure 68: Processing steps of automatic labeling algorithm, based on semantic segmentation LiDAR.

New method generates precise bounding boxes for the object of interest. The issue with imprecise labels is fixed. Examples in figure 68 depict simple cases when all pedestrians are clearly visible and separated. Useful information is provided by point ID. If two pedestrians are overlapping, ID is the only information which makes is possible to determine which point is for which pedestrians. Similarly to the one created for real Waymo data, the proposed algorithm filters out some of the bounding boxes. Filtering works similarly to the one created for the real sensor data. If for pedestrians bounding boxes the value of IoU is higher than 0.7, the further box is removed. On the range image only closer pedestrian is visible. The second type of filtering is based on the amount of points which created the box. For example only hand is sticking out from around the corner, it would be hard for detector to detect only hand as a pedestrian. If box is created out of a few points, it is just filtered out from the output. Figure 69 depicts example of the correct pedestrian labeling.

Figure 69: Processing steps of automatic labeling algorithm, based on semantic segmentation LiDAR.

Recording with precise automatic labeling can be done even with moving pedestrians and driving a car which handles the LiDAR sensor. Simulation does not have to be slowed down or stopped, like it was done in comparable published solutions [8]. Data gathering works in real-time on simulation.

6.6.4 Verification of synthetic data improvements

To determine, if there is an progress in terms of synthetic data quality and automatic labeling, an additional detector was trained. First detector validated on the synthetic data indicated over 81% mAP. Improved automatic labeling method and data quality should increase the result of the detector. Figure 70 plots the precision, recall and F1-score of detector trained on the improved synthetic data.

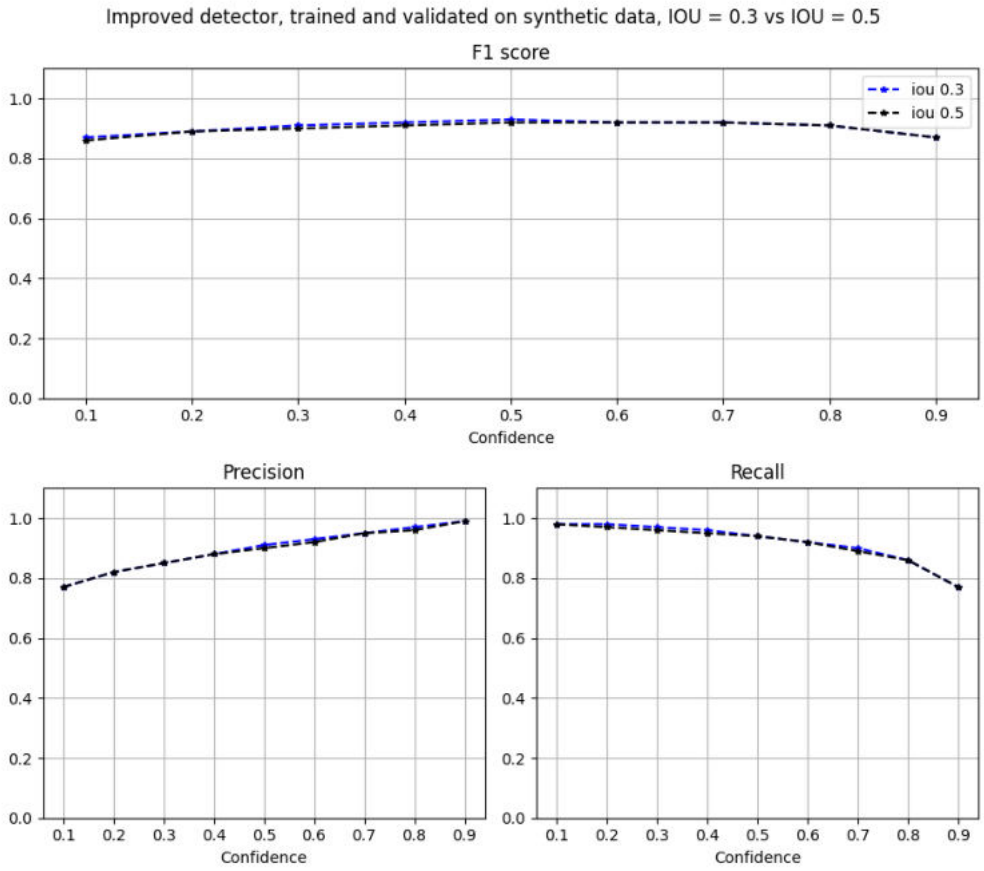

Figure 70: Pedestrian detector performance charts for precision, recall and F1-score. Detector trained and validated on synthetic data.

Detector trained on the improved synthetic data works significantly better than the first one. The highest F1-score is 0.93 and there is almost no change in its value for a wide range of confidence thresholds. Plot in figure 71 depicts precision-recall curve with mAP values achieved for IoU = 0.3 and 0.5.

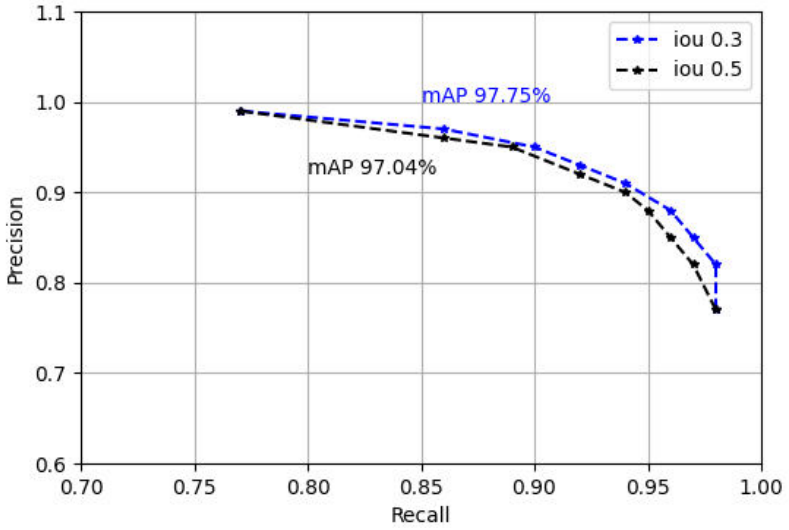

Figure 71: Precision-recall curve for pedestrian detector, with mAP values for IoU = 0.3 and 0.5 indicated. Detector trained and validated on synthetic data.

Detector indicates mAP value over 97%, which is close to perfection. Changes introduced to the generation method and automatic labeling algorithm, resulted in the expected improvements in the results. Nevertheless, the most important is to answer the question, if those improvements are going to be visible on detector validated on the real Waymo data. The next experiment aimed at calculating the performance charts for detector trained only on the synthetic data but validated on the real sensor data. Figure 72 depicts plots of precision, recall and F1-score for this experiment. Plot compares the performance of the same detector for the real and synthetic validation data.

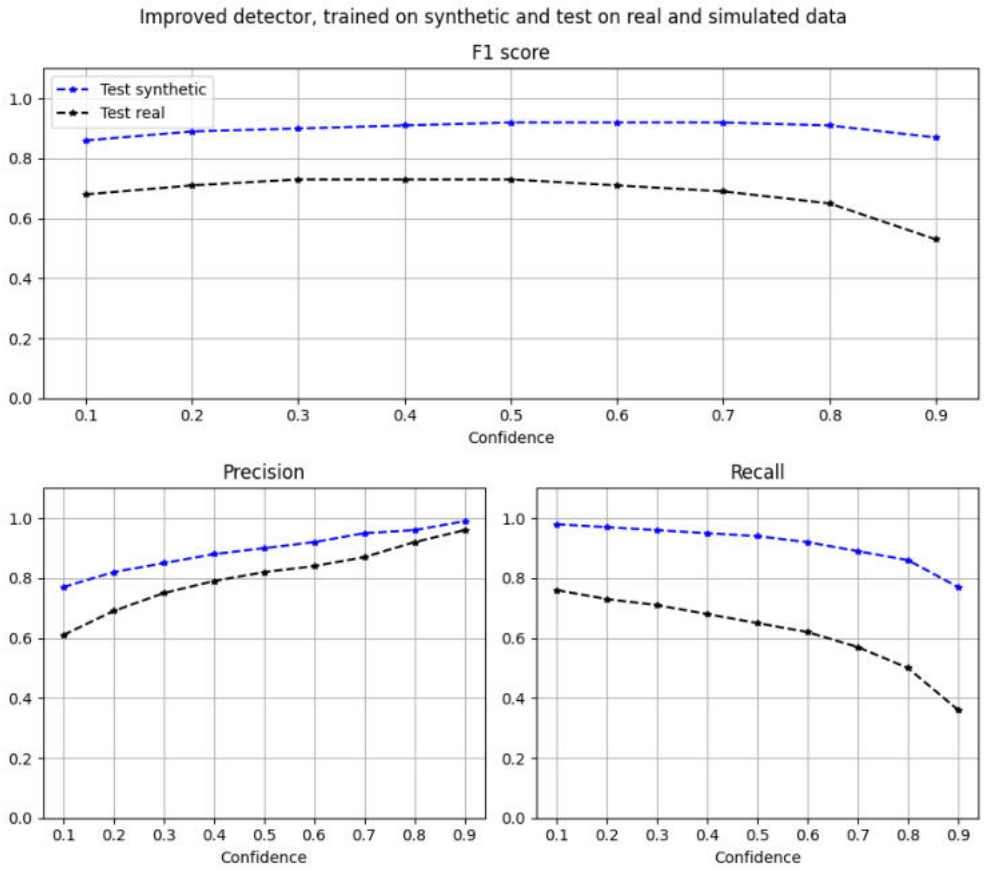

Figure 72: Pedestrian detector performance charts for precision, recall and F1-score. Comparison of detector performance calculated on synthetic and real data, IoU 0.5.

Detector validated on the real Waymo data indicates significantly lower performance than the same detector validated on the synthetic data. In comparison to the first detector, which totally failed to work on real data (F1-score < 0.4), F1-score = 0.76 is a great improvement. Figure 73 depicts the plot of precision-recall curve with mAP values indicated.

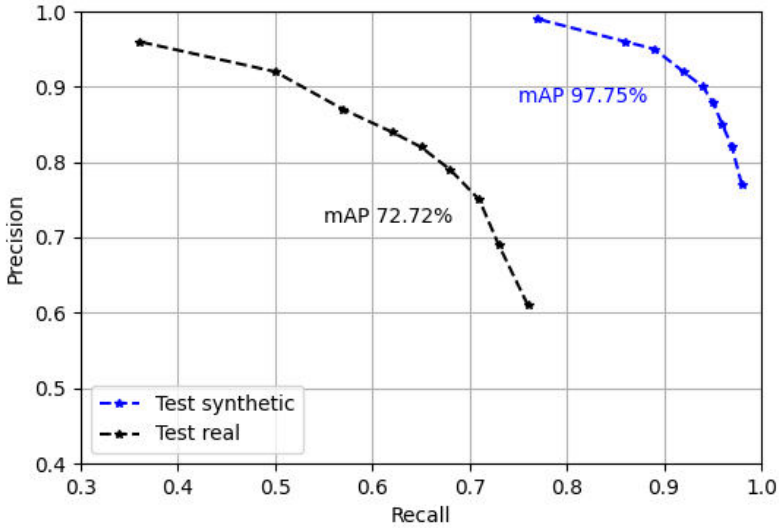

Figure 73: Precision-recall curve for pedestrian detector, with mAP values. Comparison of detector performance calculated on synthetic and real data, IoU 0.5.

Detector trained with improved synthetic data indicates much higher values of mAP, in comparison to the first try. So big difference between the synthetic and real data validation might be related to the difficulty of validation dataset. Even with great detector, difficult scenarios may indicate low performance of the detector. The proper evaluation of detector has to be done by performance comparison to the detector trained only on the real data. The results of such comparison are descried in the next chapter of this work.

To compare all already trained and evaluated detectors, table 8 is created. Table provides the summary concerning performance of each detector validated on the real and synthetic data.

Table 8: Summary results for pedestrian detector trained on synthetic data and validated on real or synthetic data.

Model	Size of training data	precision	recall	F1-score	mAP %
F-S IoU=0.5, Test: S	1.2k	0.80	0.76	0.78	81.18
F-S IoU=0.3, Test: S	1.2k	0.88	0.83	0.85	88.53
F-S IoU=0.5, Test: R	1.2k	0.34	0.26	0.29	18.27
F-S IoU=0.3, Test: R	1.2k	0.43	0.34	0.38	29.7
I-S IoU=0.5, Test: S	2.5k	0.92	0.92	0.92	97.04
I-S IoU=0.3, Test: S	2.5k	0.91	0.94	0.93	97.75
I-S IoU=0.5, Test: R	2.5k	0.79	0.68	0.73	72.72
I-S IoU=0.3, Test: R	2.5k	0.82	0.70	**0.76**	**78.17**

where:
$F - S$ => First detector trained on synthetic data
$I - S$ => Second detector trained on improved synthetic data
$Test : S$ => Detector validated on synthetic data
$Test : R$ => Detector validated on real data

To see that detector, which did not seen any real sensor data works, the figure 74 with example predictions is provided.

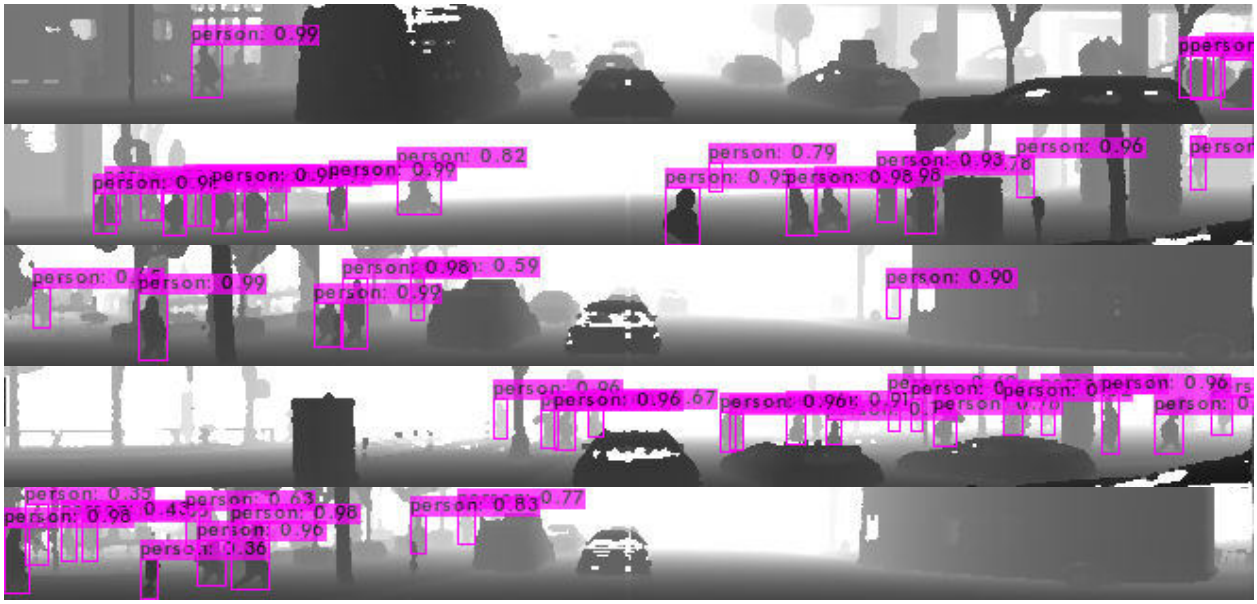

Figure 74: Example prediction of pedestrians detector trained on synthetic data. Predictions are done on the real LiDAR sensor data.

Those predictions are done on real validation data generated out of the Waymo dataset.

6.7 Comparison of training results for different size of training datasets

The goal of the research is to determine if the synthetic data can substitute real data for supervised learning, or at least to improve model performance by increasing the quality of training data. To estimate if simulation data is valuable, performance of detector trained on the real dataset has to be compared to the one trained on the synthetic data. The first step is to perform training with the usage of the real Waymo data only. Size of the training dataset is exactly the same as the one for synthetic data: 2.5k of range images. Validation dataset is the same one, and includes around 600 frames. Plot in figure 75 depicts the model training on the real data.

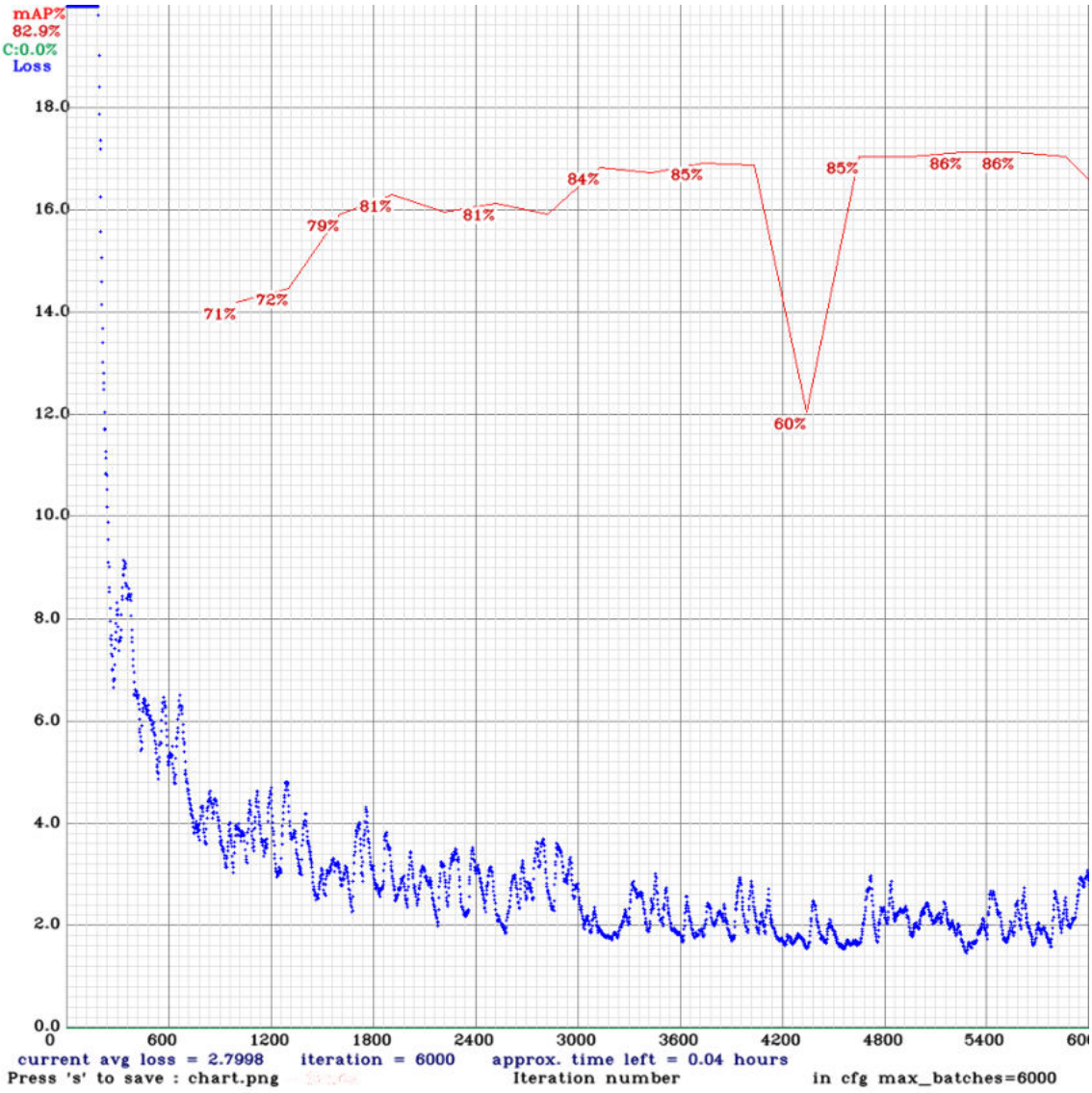

Figure 75: Training progress chart, for YOLOv4 pedestrian detector. Model trained and validated on real sensor data.

As it was expected, the training performed with the usage of the real LiDAR data only reached higher mAP than the one trained on synthetic data. Nevertheless, considering detector trained on the synthetic data with IoU = 0.3, the difference is less than 8% of mAP value. That difference may be related to the fact that the simulated scenarios are less diverse than the scenarios for the real sensor data. The second factor is that simulated sensor might be slightly in different position than in the real vehicle, which as it was already shown, can have significant impact on the obtained results. Before formulating the final judgment concerning the synthetic data, a few more experiments were performed. In the next experiment, the synthetic data and the real data were mixed and used as a new training data. In such a way the size of training data could be increased without adding and new real sensor data. Training process for such an experiment was identical as for all other experiments. The only difference consisted in the fact that the already generated synthetic data 2.5k and real data 2.5k were merged together to build twice bigger training dataset.

The result of this experiment was compared with the performance of two other detectors trained already on only real and only synthetic data. Figure 76 depicts the precision, recall and F1-score for all three detectors, validated on the same real sensor data.

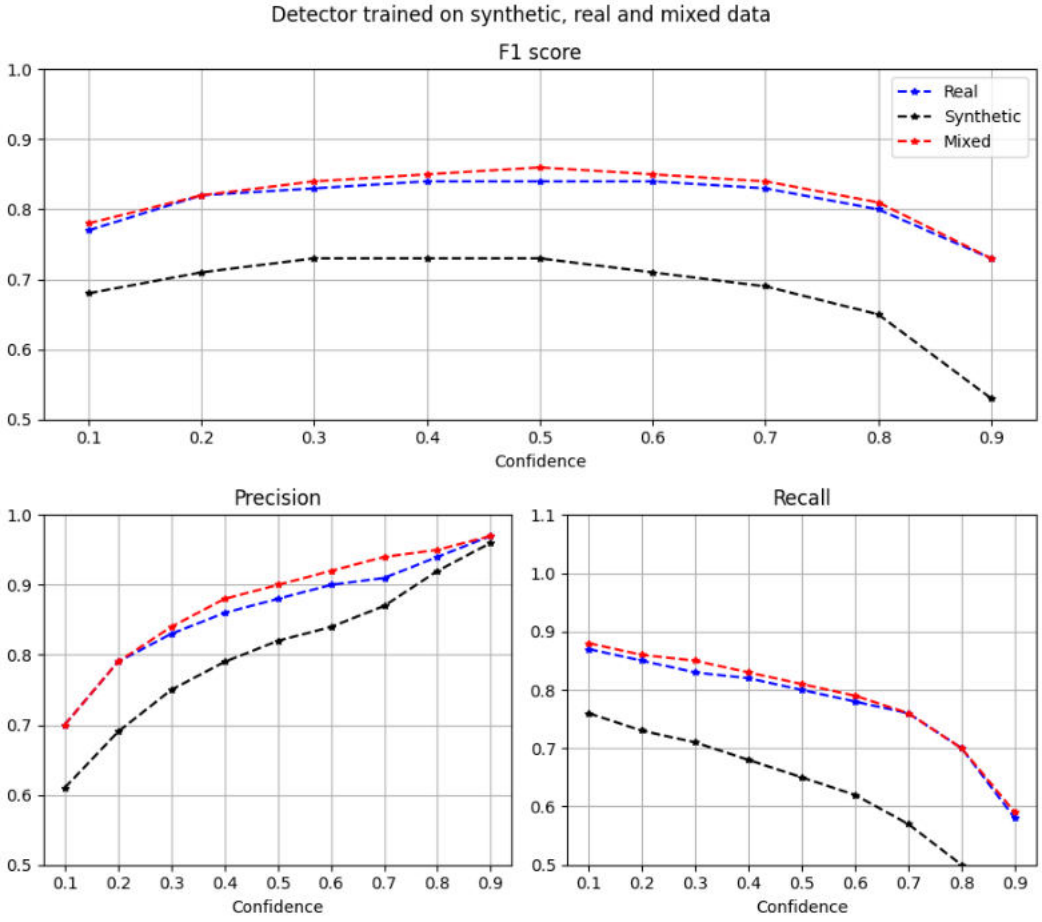

Figure 76: Pedestrian detector performance charts for precision, recall and F1-score. Comparison of three detectors, trained on real, synthetic and mixed data. IoU = 0.5

Detectors performance proved the success of the assumed approach. Model trained with the mixed data, outperformed the one trained only on the real data. For all values of the confidence threshold model shows higher value of the F1-score. The F1-score is only 2% higher, but it is higher. None of additional real sensor recording was used to improve the performance of the detector. What's interesting, the highest difference appears for model precision. Recall is almost the same as for detector trained on the real data only. Figure 77 depicts the chart of precision-recall curve for all three detectors.

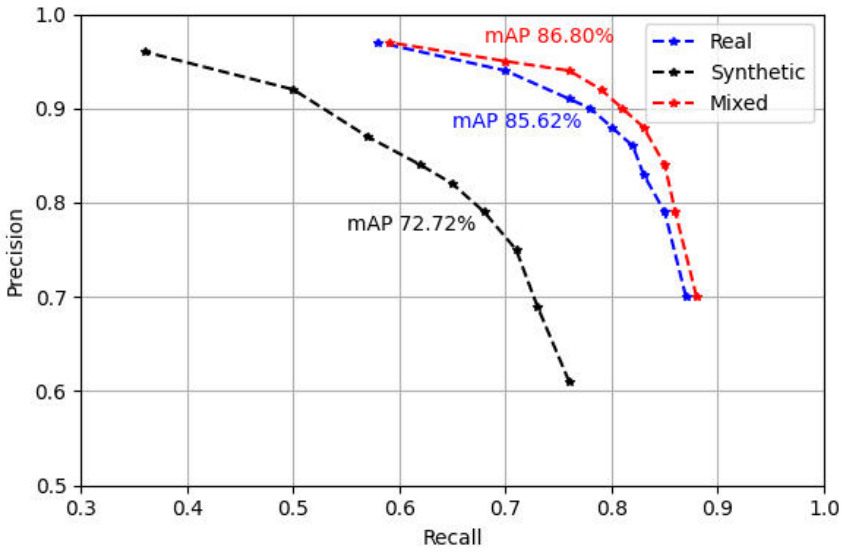

Figure 77: Precision-recall curve for pedestrian detector, with mAP values. Comparison of three detectors trained on real, synthetic and mixed data, IoU 0.5.

Detector trained on the mixed data, indicates 1.18% higher mAP than detector trained only on real data. The quality of the synthetic data, which is half of the training dataset, was high enough to allow supervised learning algorithm to improve the performance on pedestrians detection. Last experiment performed for this research answered the further question: how good would be the detector, trained on twice bigger dataset. Mixed dataset included twice more data. This experiment will indicate, the result of detector training on the same size of data, but only real data is taken into the dataset. To perform similar comparison of the models also mixed dataset is extended with the new real and synthetic data. Figure 78 depicts the results of the last experiment where training datasets were extended.

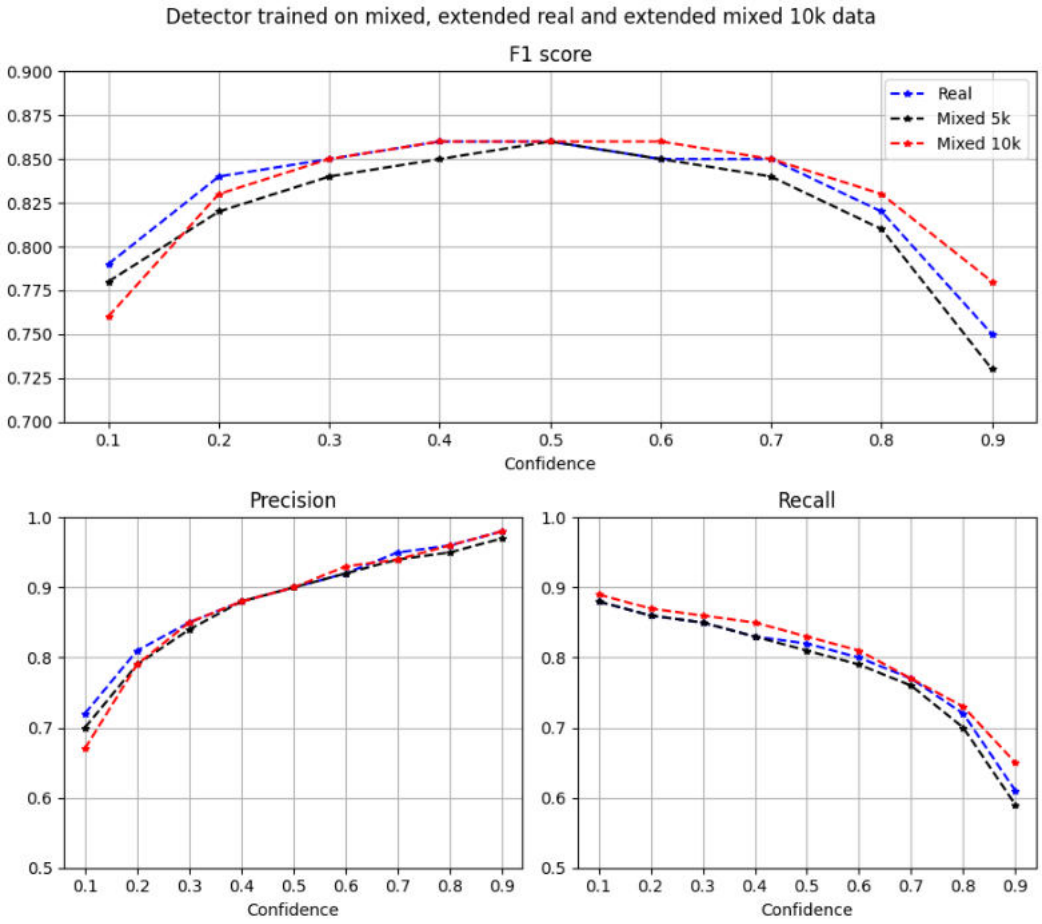

Figure 78: Pedestrian detector performance charts for precision, recall and F1-score. Comparison of three detectors, trained on real, synthetic and mixed data with doubled size of the training datasets. IoU = 0.5

Statistics of the detectors performance depicted in figure 78, shows similar situation as for the previous experiment. Only the differences between models are smaller. Application of the extended mixed dataset results in better detector performance than in case of detector trained on the real dataset only. What is more interesting, extended detector trained only with real data has just a bit better performance as the same size mixed '5k' dataset. The overview of the detectors performance is illustrated by the precision-recall curve, which is depicted in figure 79.

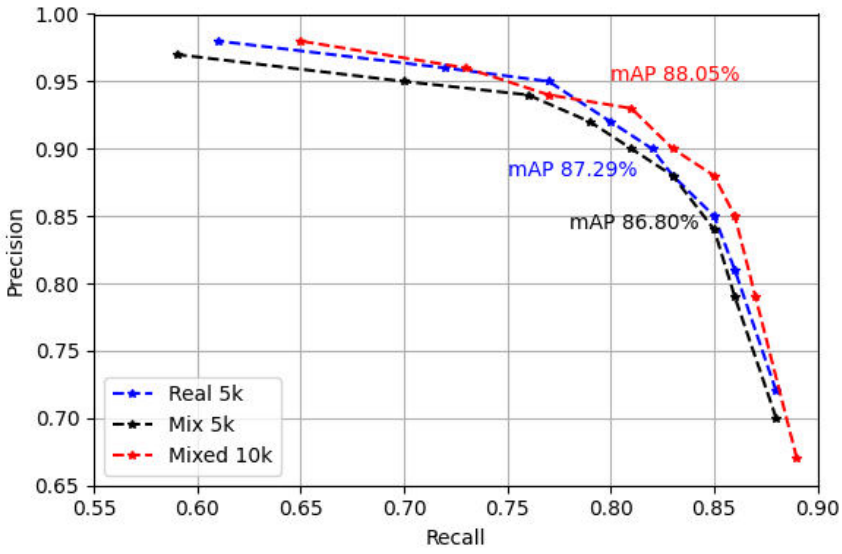

Figure 79: Precision-recall curve for pedestrian detector, with mAP values. Comparison of three detectors trained on real, synthetic and mixed data, with doubled size of the training datasets, IoU 0.5.

The differences between three models performance depicted in figure 79 are just a bit more than 1%. Such results indicate that it is hard to increase the amount of correctly detected pedestrians on the validation dataset. The difference between model consists mostly in the labeling accuracy and the number of true positives. Detector trained on the biggest dataset, shows increase in the performance for high values of confidence threshold only. Additional training data increases the values of confidence threshold, where model performs best. To indicate that difference by numbers, bar graph is provided in figure 80. The graph compares the number of TP, FP and FN for three considered detectors.

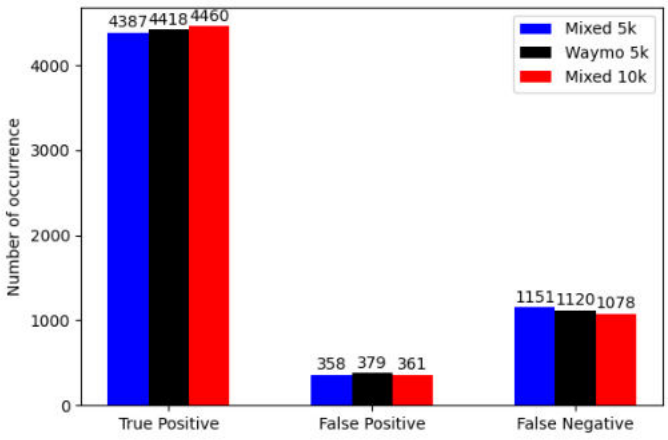

Figure 80: Bar graph of TP, FP and FN for three detectors trained on real and mixed synthetic-real data. Statistics calculated for IoU =0.5 and confidence threshold =0.6.

Performed experiments prove that the synthetic data can be successfully used to improve the

quality of the supervised learning models. Improvement indicates that the simulated LiDAR data, generated from Carla simulation, has the quality high enough to be a valuable source of data for perception algorithms training. The gap between synthetic and real data is still visible, but it is small and after mixing this data with real data it becomes possible to improve the model performance. To summarize the results of trained detectors, table 9 was prepared.

Table 9: Summary results for pedestrian detector trained on synthetic,real and mixed data.

Model	Size of training data	precision	recall	F1-score	mAP %
Synth, IoU=0.5	2.5k	0.79	0.68	0.73	72.72
Synth, IoU=0.3	2.5k	0.82	0.70	0.76	78.17
Real, IoU=0.5	2.5k	0.86	0.82	0.84	85.62
Real, IoU=0.3	2.5k	0.86	0.86	0.86	89.93
Mixed, IoU=0.5	5k, 50/50	0.90	0.81	0.86	86.80
Mixed, IoU=0.3	5k, 50/50	0.92	0.83	0.87	90.87
Real, IoU=0.5	5k	0.90	0.82	0.86	87.29
Real, IoU=0.3	5k	0.90	0.85	0.88	**91.52**
Mixed, IoU=0.5	10k, 30/70	0.90	0.83	0.86	**88.05**
Mixed, IoU=0.3	10k, 30/70	0.92	0.84	0.88	91.27

For all models, performance marked in table 9 is evaluated for the same real sensor validation dataset.

6.8 Synthetic LiDAR data limitation

The method of synthetic LiDAR data generation proposed in this research has a few major limitations. There are particular situations where the method cannot deliver valuable data for the task. That limitations are mostly related to the fact that CARLA version 0.9.12 does not support reflective generation for the LiDAR sensor point cloud. Good example of task that can't be performed on such synthetic data, is line detection, which is mostly based on the reflectivity difference between road tarmac and line painting [75, 76]. Without simulation support on material reflectivity, there is no chance to generate useful data for algorithms. There are plans defined by Carla simulation developer to improve the model and add this properties of the point cloud to improve the sensor model simulation. Right now the only possibility would be to use the semantic segmentation tag of the object to mimic the value of reflectivity for each point. But such implementation is just a simplification of the reflectivity model. Except of LiDAR reflectivity measurement the real sensors can work also in dual return mode. In such mode the sensor returns not only one point cloud out of the first ray cast hit, but also the second one. Dual return mode allows to gather more information

about scanned environment. None of simulators such as Carla can simulate LiDAR model with dual return mode.

Second important limitation of the proposed method is the amount of diverse scenarios available out of the box in the Carla simulator. In case of the real dataset gathering and labeling, some part of time saved out of automatic generation and labeling of synthetic data has to be put into the scenario creation. In case of Carla simulator there are almost infinite possibilities to create road scenarios. The main concern is that each scenario has to be developed in graphical editor. This process, even for the experienced developer, may be time consuming. Carla out of the box includes only 7 different maps, which are not so big, in comparison to the newest virtual worlds created for the game industry. This poses an limitation in case of diverse road scenarios generation. In this research all 7 maps where people can walk around were used, and this allowed to generate over 5k of range images. With randomized traffic set on roads and sidewalks, it would be possible to create more data but not significantly more. In comparison to popular in scientific works GTA V, Carlas maps are tiny. For example in publication [9], MOTsynth dataset was generated out of virtual GTA V world and included over 1.3 million of images. It was possible to generate such a big, if not the biggest, synthetic dataset with automatically labeled objects, like pedestrians since the simulator had huge virtual world already available. Work with scenario creation was already done by the game developers. There are already some software solutions to automatically generate new road scenarios. Good examples of such solutions are ASAM OpenSCENARIO [77] or OpenDrive [78], which are an open standard to describe the content of driving and traffic. They allow to record the driving scenarios from the real roads and then use them to automatically render the same environment in the simulation. Carla is supporting such files to enable fast scenario generation. Still this solution has limitation in the amount of details stored in the recording. Mostly only simple road shapes or crossings are saved with some traffic signs. All the surrounding of the road is missing in the rendered scenario. Figure 81 depicts the example of an OpenDrive map rendered in Carla simulator.

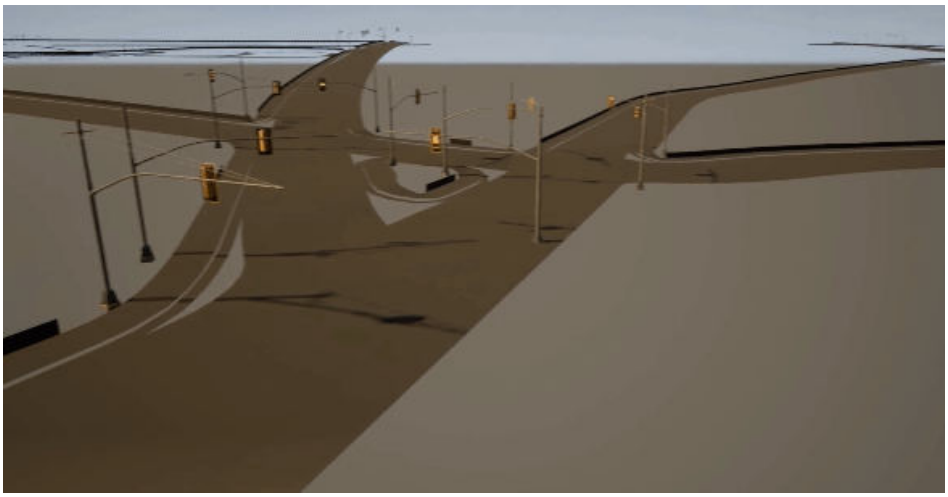

Figure 81: Example map rendered out of the OpenDrive file standard, in Carla simulator.

Road shape and some traffic signs are rendered automatically, but the overall view of the scenario is empty. This solution may help with scenario generation but does not make it fully automated.

7 Summary and final conclusion

Automotive industry is aiming to deliver fully autonomous vehicles to the customers. The first company, which will be able to achieve this goal, will change the people perception of the everyday traveling. Nowadays, a lot of effort has to be put into the development of technology, which may achieve such a high level of vehicle driving automation. To perform fully autonomous drive on every road and weather conditions, already known technology must be fairly improved. The main challenge is to set up a sensing system, which allows to perfectly understand the road traffic and movement of each participant. Plenty types of sensors are used for this perception tasks and most of them relay on the achievements of the machine learning technology. Supervised learning technique requires huge amount of labeled data to train models which detect, classify and track the road participants. The goal of this research was to propose the solution for the problem of training datasets gathering. Research aim was to generate the auto labeled synthetic data, which can substitute the real sensor data. This approach may significantly reduce the development time of the new innovative perception models. What's more, this method can significantly reduce the expenses on the research projects. Vehicle fuel, track driving time and other expenses can be lowered, by transferring the data gathering process onto the computer simulation. The focus of this research was put on evaluation of the simulated data usability, in terms of advanced driving assistance systems. The feasibility study on the modern 3D rendering simulations was experimentally verified, by performing several experiments on the real sensors and ADAS functionalities. The first step was to determine the usability of such simulation data in terms of ADAS testing. Most common sensor used for such functionalities is camera. This sensor was the first choice to be tested on the simulated data. One of the auxiliary thesis of this work states that simulation data can be treated as a valid source of input for ADAS, if the results of the same task are constant and repeatable. To verify this statement concerning simulation data usefulness for a camera sensor, a few experiments were set up and performed. The first step was to set up experiment where the controlled simulated road scenario was repeatedly shown to the sensor. As the source of the synthetic scenario the Carla simulator was used. Prepared scenario included two speed limit signs, which were detected by the traffic sign recognition functionality of the real ADAS sensor (Mobilieye 6). Constant and repeatable results of over 400 correct sign detections, gave the answer, that such a 3D simulated road scenario can be used as a valid input for TSR ADAS functionality. Experiment proved the real-time abilities of the developed hardware-in-the-loop (HiL) system. It is obvious that this single experiment couldn't prove that the proposed simulation data has quality high enough to be a suitable substitution of the real road data. This type of TSR functionality testing has been already described in the scientific papers. To push already known technology to the next level, more complex closed-loop HiL experiment was carried out. This experiment was to determine if the synthetic simulation data can be used as an input for the lane keeping algorithm. The goal of the experiment was to put a virtual vehicle to be controlled by the real camera sensor. Lane detection done by the sensor was used as the input for the lane keeping algorithm, which controls the virtual vehicle to drive along the lane. The detection and control loop keeps simulated vehicle on the road. The experiment is the first published hardware in the closed loop test, where camera sensor, without the whole vehicle, was used [40]. Experiment results (chapter 4.5.2) show that an artificial on fly generated 3D camera image is a valuable source of the data for the ADAS functionality, which was developed with the usage of real road data only. Synthetic data quality was good enough to test the performance of the lane keeping algorithm without the necessity to drive any car on the track or real road.

ADAS functionalities are not only supported by the camera sensor data. There are variety of more or less useful sensor types, which generate necessary data for a vehicle road perception. One of the newest types of sensors is a multiple channel LiDAR sensor. This sensor, which becomes less and less expensive, delivers high precision spatial information about scanned environment. A lot of advantages against 2D camera images makes the LiDAR interesting and promising type of sensor. Due to the arguments mentioned above, also the LiDAR sensor simulation was validated against usability on the real ADAS functions. Proposed Carla simulator, is able to generate synthetic point cloud like the real LiDAR sensor. Those abilities were used to test the usefulness of point cloud in the real vehicle application. Performed and described experiment included real-time substitution of LiDAR data by artificial data, generated by Carla simulator. The goal of the experiment was to run a free space detection algorithm on the real and synthetic data. The performance of the detection algorithm for both types of data was compared. Discussed experiment was the first conducted synthetic LiDAR injection to the ADAS algorithm, which works identical as on the real data. Results of the synthetic LiDAR data experiments were published in Sensors journal in 2021 [40].

The further research was aimed at verification of the second auxiliary thesis, which states that the synthetic data can be a valuable source of input data for machine learning algorithms, if those trained with the usage of this artificial data indicate higher performance than the algorithms trained on the real data only. The idea behind such an approach is simple. If synthetic data quality and characteristic is the same as for the real data, deep neural network model should not indicate any difference on its performance. The possibility to substitute even part of the real dataset by synthetically generated data gives huge advantages. Supervised learning models require vast amounts of data. Data recording is one part of the job. Second is to label the objects of interest. Synthetic data would be a perfect solution to speed up and lower the cost of development of vehicles perception systems. The main problem results from the fact that the 3D rendering engines mostly do not generate the data of the sufficient quality, which could be successfully combined with the real data. This phenomenon is called the synthetic to real gap. If the model trained on synthetic data indicates much lower performance than the model trained on real data, it mean that there is a gap between real and synthetic data. Experiment goal for this research was simple, train a detector with the usage of synthetic data and compare its performance with the performance of the detector trained on the real data only. If thesis of the work is true, it should be possible to improve the neural network model by the usage of generated artificial data. Improvement should be possible, for example by increased amount of training data. Diverse simulated road scenarios were the source of the synthetic data and the Waymo dataset was the source of the real sensor data. YOLOv4 deep neural model was trained to detect pedestrians out of the road scenarios. Input for the model was the range image generated out of the LiDAR sensor point cloud. The same data preparation was performed to create synthetic and real datasets. Detectors were trained on those custom datasets of different sizes. Model trained with the usage of synthetic data only, performed around 10% worse than the one trained on real data. Those results show that there is still a gap between simulated LiDAR data and the real one. Nevertheless, those synthetic data mixed with real data performed better than real dataset only. This improvement proves that those type of 3D rendering engines is capable of generating high quality data to be used as a valuable source for the supervised learning technology in vehicles perception systems. Moreover, achieved performance of the pedestrian detector outperformed models trained with Carla simulator, which were described in the published papers [38, 35]. The detector performance reached state of the art mAP value (88.05%) for LiDAR sensor detector based on the 2D projection of the point cloud [8, 79, 80].

Developed hardware and software in this work includes solutions for:

- Hardware in the loop framework for ADAS validation supported by high fidelity 3D rendering engine. Framework supports the real-time measurements and open/closed loop testing of ADAS functionalities. Measurement abilities were proven on the traffic sign detection test, performed on the aftermarket Mobileye 660 camera. Closed loop abilities of the framework were validated on the lane keeping algorithm, which is the first successful experiment of this type, which does not require vehicle for algorithm validation in real-time.
- Adaptation of open source Carla simulator for the purpose of ADAS functionality testing and training. Developed of .net interface for simulator to support data generation and injection on Windows operating system. Simulator integrated with the testing Morphee core, proved its value in the several experiments conducted for camera and LiDAR sensors.
- Lane keeping algorithm based on the third degree polynomial line model detected by camera sensor. Algorithm allows to control virtual vehicle steering to keep it on track.
- Validation of the free space detection algorithm on synthetic LiDAR data. Experimentally verified solution shows that the simulated LiDAR model may be successfully used to test free space detection algorithm. Simulation run on the testing framework can be a valid substitution of the real vehicle test drives on the road.
- Automatic labelling method for 3D objects on camera images and LiDAR range images, generated out of Carla simulation. Novel method based on the semantic segmentation point cloud data precisely calculates position of the bounding boxes for objects placed in the simulation world relative to the sensor location.
- Generation of the configurable range images out of the Carla simulation. Generation method allows to created identical to the real Waymo LiDAR sensor data in terms of quality and characteristic.
- Pedestrian detection functionality for LiDAR sensor data, ready to be applied to the real sensor. Trained detector is capable of detecting pedestrians out of 64-channel LiDAR sensor with average precision of 90%.

Results of the performed experiments allow to draw the following conclusions:

- Simulation of the road scenario based on the Unreal Engine 4, such as Carla, is able to generate the data of the quality high enough to be used as an input for the validation of Advance Driving Assistant Systems. Experiments shows that tested functionality, for example TSR, on synthetic data outputs constant and repeatable results.
- Proposed solution, where artificial images are the input for ADAS, may be used to reduce the amount of work and expenses put into development of novel perception systems in automotive industry.
- Modern simulator Carla, generates data which may be used as a custom input dataset for the deep neural network models, to improve the performance of the detector.

- LiDAR sensor model, provided by Carla simulation, can improve the performance of the pedestrian detectors by increasing the amount of training data. Experiments proved that the gap between synthetic and real sensor data can be bridged, with the usage of modern high fidelity 3D rendering engines.

Assuming the drawn conclusions, main thesis of this work can be confirmed.

8 Bibliography

References

[1] SAE International: Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems, Standard J3016, January 2014.

[2] Clausen, J., Olteanu, Y. New players in the automotive industry: Waymo, Build Your Dreams and Sono Motors. Technical report, 2021.

[3] Sun, Pei , Kretzschmar, Henrik , Dotiwalla, Xerxes , Chouard, Aurelien , Patnaik, Vijaysai , Tsui, Paul , Guo, James , Zhou, Yin , Chai, Yuning , Caine, Benjamin , Vasudevan, Vijay , Han, Wei Ngiam, Jiquan Zhao, Hang Timofeev, Aleksei Ettinger, Scott Krivokon, Maxim Gao, Amy Joshi, Aditya Anguelov, Dragomir. (2020). Scalability in Perception for Autonomous Driving: Waymo Open Dataset. 2443-2451. 10.1109/CVPR42600.2020.00252.

[4] Geiger, Andreas Lenz, P Stiller, Christoph Urtasun, Raquel. (2013). Vision meets robotics: the KITTI dataset. The International Journal of Robotics Research. 32. 1231-1237. 10.1177/0278364913491297.

[5] Caesar, Holger Bankiti, Varun Kumar Reddy Lang, Alex Vora, Sourabh Liong, Venice Erin Xu, Qiang Krishnan, Anush Pan, Yu Baldan, Giancarlo Beijbom, Oscar. (2020). nuScenes: A Multimodal Dataset for Autonomous Driving. 11618-11628. 10.1109/CVPR42600.2020.01164.

[6] Verhaegen, Michel Development, Gietelink, Olaf Ploeg, Jeroen De Schutter, Bart. (2006). Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations. Vehicle System Dynamics. 44. 569-590. 10.1080/00423110600563338.

[7] Ros, G., Sellart, L., Materzynska, J., Vazquez, D., Lopez, A.M. The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes. Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 3234-3243.

[8] Hurl, B., Czarnecki, K., Waslander, S. Precise Synthetic Image and LiDAR (PreSIL) Dataset for Autonomous Vehicle Per-ception. In 2019 IEEE Intelligent Vehicles Symposium (IV), 2019, pp. 2522-2529.

[9] Fabbri, M., Brasó, G., Maugeri, G., Cetintas, O., Gasparini, R., Ošep, A., Calderara, S., Leal-Taixé, L., Cucchiara, R. Motsynth: How can synthetic data help pedestrian detection and tracking?. Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 10849-10859.

[10] Dahun, Kim Woo, Sanghyun Lee, Joon-Young Kweon, Inso. (2020). Video Panoptic Segmentation. 9856-9865. 10.1109/CVPR42600.2020.00988.

[11] Fabbri, Matteo Lanzi, Fabio Calderara, Simone Palazzi, Andrea Vezzani, Roberto Cucchiara, Rita. (2018). Learning to Detect and Track Visible and Occluded Body Joints in a Virtual World.

[12] Galko, Clément Rossi, Romain Savatier, Xavier Payá-Vayá, Guillermo Blume, Holger. (2017). Vehicle Hardware-In-the-Loop System for ADAS Virtual Testing.

[13] Bhadani, Rahul Sprinkle, Jonathan Bunting, Matthew. (2018). The CAT Vehicle Testbed: A Simulator with Hardware in the Loop for Autonomous Vehicle Applications. Electronic Proceedings in Theoretical Computer Science. 269. 32-47. 10.4204/EPTCS.269.4.

[14] Hwang, Taehun Roh, Jihoon Park, Kihong Hwang, Jeongho Lee, Kyu Lee, Kangwon Lee, Soo-Jin Kim, Young-Jun. (2006). Development of HILS Systems for Active Brake Control Systems. 4404 - 4408. 10.1109/SICE.2006.314663.

[15] Zhou, Jinwei Schmied, Roman Sandalek, Alexander Kokal, Helmut Re, Luigi. (2016). A Framework for Virtual Testing of ADAS. SAE International Journal of Passenger Cars - Electronic and Electrical Systems. 9. 10.4271/2016-01-0049.

[16] Solmaz, S. Holzinger, Franz. (2019). A Novel Testbench for Development, Calibration and Functional Testing of ADAS/AD Functions. 10.1109/ICCVE45908.2019.8965225.

[17] Mohammed, Abdul Sajeed Amamou, Ali kloutse ayevide, Follivi Kelouwani, Sousso Agbossou, K. Zioui, Nadjet. (2020). The Perception System of Intelligent Ground Vehicles in All Weather Conditions: A Systematic Literature Review. Sensors. 20. 10.3390/s20226532.

[18] Velodyne VPL-16 , Versatile real-time lidar sensor. 2019.

[19] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V. CARLA: An Open Urban Driving Simulator. CoRL , vol. 78 of Proceedings of Machine Learning Research, 2017, pp. 1-16.

[20] FEV Europe GmbH: MORPHEE - Automation System 2017

[21] Wu T., Fu H., Liu B., Xue H., Ren R., Tu Z. Detailed Analysis on Generating the Range Image for LiDAR Point Cloud Processing. Electronics 2021, 10, 1224. https://doi.org/10.3390/electronics10111224

[22] Bochkovskiy, Alexey Wang, Chien-Yao Liao, Hong-yuan. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection.

[23] Paranjape, Ishaan Jawad, Abdul Xu, Yanwen Song, Asiiah Whitehead, Jim. (2020). A Modular Architecture for Procedural Generation of Towns, Intersections and Scenarios for Testing Autonomous Vehicles. 162-168. 10.1109/IV47402.2020.9304625.

[24] Prescan: Simulation of adas active safety. https://www.plm.automation.siemens.com/global/en/products/simulation-test/active-safety-system-simulation.html. Web page, accessed 17 October 2022.

[25] Ziegler, Stephan Höpler, Robert. (2011). Extending the IPG CarMaker by FMI Compliant Units. 10.3384/ecp11063779.

[26] rFpro: Simulation software. https://rfpro.com/about-events-automotive/. Web page, accessed 17 October 2022.

[27] Mechanical Simulation Corporation, CarSIM. https://www.carsim.com/. Web page, accessed 17 October 2022.

[28] NVIDIA Drive - autonomous vehicle development platforms,https://developer.nvidia.com/drive/drive-constellation, web page, accessed 17 October 2022.

[29] Microsoft Airsim. https://microsoft.github.io/AirSim/. web page, accessed 17 October 2022.

[30] LGSVL Simulator,https://www.svlsimulator.com/product/simulation/, web page, accessed 17 October 2022.

[31] Nvidia Drive Sim: https://www.youtube.com/watch?v=DXsLDyiONV4, Web page, accessed 17 October 2022.

[32] Galko, Clément Rossi, Romain Savatier, Xavier Payá-Vayá, Guillermo Blume, Holger. (2017). Vehicle Hardware-In-the- Loop System for ADAS Virtual Testing.

[33] Kassem, N.S.; Masoud, A.M.; Aly, A.M.B. Driver-in-the-Loop for computer-vision based ADAS testing. In Proceedings of the 2021 3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES), Giza, Egypt, 23–25 October 2021; pp. 252–256. https://doi.org/10.1109/NILES53778.2021.9600491.

[34] Tremblay, Jonathan Prakash, Aayush Acuna, David Brophy, Mark Jampani, Varun Anil, Cem To, Thang Cameracci, Eric Boochoon, Shaad Birchfield, Stan. (2018). Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization. 1082-10828. 10.1109/CVPRW.2018.00143.

[35] Wang, F., Zhuang, Y., Gu H., Hu, H. Automatic Generation of Synthetic LiDAR Point Clouds for 3-D Data Analysis. IEEE Transactions on Instrumentation and Measurement, 2019, pp. 1-3. DOI: 10.1109/TIM.2019.2906416.

[36] Hurl, Braden Czarnecki, Krzysztof Waslander, Steven. (2019). Precise Synthetic Image and LiDAR (PreSIL) Dataset for Autonomous Vehicle Perception. 2522-2529. 10.1109/IVS.2019.8813809.

[37] Spiegel, Steven Chen, Jorge. (2021). Using Simulation Data From Gaming Environments for Training a Deep Learning Algorithm on 3D Point Clouds. VIII-4/W2-2021. 67-74. 10.5194/isprs-annals-VIII-4-W2-2021-67-2021.

[38] Dworak, D., Ciepiela, F., Derbisz, J., Izzat, I., Komorkiewicz, M., Wojcik, M. Performance of LiDAR object detection deep learning architectures based on artificially generated point cloud data from CARLA simulator. International conference on Methods and Models in Automation and Robotics MMAR, 2019, pp. 600-605.

[39] Interval Zero, Real-Time Operating System (RTOS) Platform Vision. Available online: https://www.intervalzero.com/products/rtos-platform-vision (accessed on 19 October 2022).

[40] Jabłoński, P., Iwaniec, J., Jabłoński M. Multisensory testing framework for advanced driver assistant systems supported by high-quality 3D simulation. Sensors 2021, 21(24), figure 2.

[41] Vector, FDX-Fast Data Exchange with CANoe. Available online: https://assets.vector.com/cms/content/know-how/application-notes/AN-AND-1-119 Fast Data Exchange with CANoe.pdf (accessed on 20 October 2021).

[42] Gao, Feng Li, Caihong Zhang, Bowen. (2021). A Dynamic Clustering Algorithm for Lidar Obstacle Detection of Autonomous Driving System. IEEE Sensors Journal. PP. 1-1. 10.1109/JSEN.2021.3118365.

[43] Nowak, T.; Ćwian, K.; Skrzypczyński, P. Real-Time Detection of Non-Stationary Objects Using Intensity Data in Automotive LiDAR SLAM. Sensors 2021, 21, 6781. https://doi.org/10.3390/s21206781

[44] Bradski, G., Kaehler, A. (2008). Learning OpenCV: Computer vision with the OpenCV library. Reilly Media, Inc.

[45] Sallab, Ahmad Sobh, Ibrahim Zahran, Mohamed Essam, Nader. (2019). LiDAR Sensor modeling and Data augmentation with GANs for Autonomous driving.

[46] Smart Vehicle FEV Project. Available online: https://smart-vehicle.fev.com/#adas (accessed on 20 October 2021).

[47] Tian, D., Han, Y., Wang, B., Guan, T., We, W. A Review of Intelligent Driving Pedestrian Detection Based on Deep Learning. Computational Intelligence and Neuroscience 2021, 2021, Article ID 5410049. https://doi.org/10.1155/2021/541004

[48] Girshick, R., Donahue, J., Darrell, T., Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 580–587.

[49] Redmon, J., Divvala, S. Girshick, R., Farhadi, A. You only look once: unified, real-time object detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 779–788.

[50] Redmon, J., Farhadi, A. Yolo9000: better, faster, stronger. Proceedings of 30th IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017.

[51] Redmon, J., Farhadi, A. Yolov3: An incremental improvement. arXiv preprint arXiv: 1804.02767, 2018.

[52] Pathak, A.R., Pandey, M., Rautaray, S. Application of deep learning for object detection. Procedia computer science 2018, 132, pp. 1706-1717.

[53] Roszyk, K., Nowicki, M.R., Skrzypczyński, P. Adopting the YOLOv4 Architecture for Low-Latency Multispectral Pedes-trian Detection in Autonomous Driving. Sensors 2022, 22, 1082. https://doi.org/10.3390/s22031082

[54] Padilla, R., Netto, S., da Silva, E. A Survey on Performance Metrics for Object-Detection Algorithms. Conference: 2020 International Conference on Systems, Signals and Image Processing (IWSSIP). 2020. DOI:10.1109/IWSSIP48289.2020

[55] Gong, M. A Novel Performance Measure for Machine Learning Classification. International Journal of Managing Information Technology, 2021, 13(1), pp. 1–19. DOI: https://doi.org/10.5121/ijmit.2021.13101

[56] Olivier, M., Verschoof-van der Vaart, W. Implementing State-of-the-Art Deep Learning Approaches for Archaeological Object Detection in Remotely-Sensed Data: The Results of Cross-Domain Collaboration. Journal of Computer Applications in Archaeology, 2021, 4(1), pp. 274–289.DOI:https://doi.org/10.5334/jcaa.78

[57] Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., Savarese, S. Generalized intersection over union: A metric and a loss for bounding box regression. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 658-666.

[58] Zamanakos, Georgios Tsochatzidis, Lazaros Amanatiadis, Angelos Pratikakis, Ioannis. (2021). A comprehensive survey of LIDAR-based 3D object detection methods with Deep learning for autonomous driving. Computers and Graphics. 99. 10.1016/j.cag.2021.07.003.

[59] Kwon, SoonSub Park, TaeHyoung. (2020). Channel-Based Network for Fast Object Detection of 3D LiDAR. Electronics. 9. 1122. 10.3390/electronics9071122.

[60] Yan, Lin Liu, Kai Belyaev, Evgeny Duan, Meiyu. (2020). RTL3D: real-time LIDAR-based 3D object detection with sparse CNN. IET Computer Vision. 14. 10.1049/iet-cvi.2019.0508.

[61] Doleck, T., Lemay, D.J., Basnet, R.B. et al. Predictive analytics in education: a comparison of deep learning frameworks. Educ Inf Technol 25, 1951–1963 (2020). https://doi.org/10.1007/s10639-019-10068-4

[62] Redmon, J. Darknet: Open Source Neural Networks in C, 2013-2016, Available online: https://pjreddie.com/darknet/ (accessed on 28 May 2022).

[63] Carla Documentation: Depth map sensor. Available online: https://carla.readthedocs.io/en/0.9.8/ref_sensors/ (accessed on 09 September 2022).

[64] Velodyne Lidar HDL-64E, Available online https://velodynelidar.com/blog/hdl-64e-lidar-sensor-retires (accessed on 09 September 2022).

[65] Solana-Cipres, C.J. Albusac, Javier Castro-Schez, Jose Benitez, Luis. (2010). Automatic object labelling for monitored environments using clustering techniques. IET Seminar Digest. 2009. 1 - 6. 10.1049/ic.2009.0260.

[66] Architecture, Forensic. (2022). Experiments in Synthetic Data. 10.1002/9781119815075.ch50.

[67] Shen, Xuyang Plested, Jo Caldwell, Sabrina Zhong, Yiran Gedeon, Tom. (2022). AMF: Adaptable Weighting Fusion with Multiple Fine-tuning for Image Classification. 10.48550/arXiv.2207.12944.

[68] Darknet: Pre trained weights yolov4.conv.137, Available online: https://github.com/AlexeyAB/darknet (accessed on 28 May 2022).

[69] Santos, Claudio Papa, João. (2022). Avoiding Overfitting: A Survey on Regularization Methods for Convolutional Neural Networks. ACM Computing Surveys. 10.1145/3510413.

[70] Biasutti, Pierre Aujol, Jean-François Brédif, Mathieu Bugeau, Aurélie. (2018). Range-Image: Incorporating Sensor Topology for Lidar Point Cloud Processing. Photogrammetric Engineering Remote Sensing. 84. 10.14358/PERS.84.6.367.

[71] Waymo dataset python package source, Available online: https://github.com/waymo-research/waymo-open-dataset (accessed on 28 May 2022).

[72] Filgueira, A. Gonzalez, Higinio Lagüela, Susana Díaz Vilariño, Lucia Arias, Pedro. (2016). Quantifying the influence of rain in LiDAR performance. Measurement. 95. 10.1016/j.measurement.2016.10.009.

[73] Waymo 3D point cloud representation image with labeled objects, Available online: https://waymo.com/static/images/dataset/format/labels-3d.jpg (accessed on 28 May 2022).

[74] Jabłoński, P.; Iwaniec, J.; Zabierowski, W. Comparison of Pedestrian Detectors for LiDAR Sensor Trained on Custom Synthetic, Real and Mixed Datasets. Sensors 2022, 22, 7014. https://doi.org/10.3390/s22187014

[75] Jung, J.; Bae, S.-H. Real-Time Road Lane Detection in Urban Areas Using LiDAR Data. Electronics 2018, 7, 276. https://doi.org/10.3390/electronics7110276

[76] Yue, Xiangyu Wu, Bichen Seshia, Sanjit Keutzer, Kurt Vincentelli, Alberto. (2018). A LiDAR Point Cloud Generator: from a Virtual World to Autonomous Driving. 458-464. 10.1145/3206025.3206080.

[77] J. -M. Jullien, C. Martel, L. Vignollet and M. Wentland, "OpenScenario: A Flexible Integrated Environment to Develop Educational Activities Based on Pedagogical Scenarios," 2009 Ninth IEEE International Conference on Advanced Learning Technologies, 2009, pp. 509-513, doi: 10.1109/ICALT.2009.24.

[78] Schwab, Benedikt Kolbe, Thomas. (2022). Validation of Parametric OpenDRIVE Road Space Models. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences. X-4-W2-2022. 10.5194/isprs-annals-X-4-W2-2022-257-2022.

[79] Chai, Yuning Sun, Pei Ngiam, Jiquan Wang, Weiyue Caine, Benjamin Vasudevan, Vijay Zhang, Xiao Anguelov, Dragomir. (2021). To the Point: Efficient 3D Object Detection in the Range Image with Graph Convolution Kernels.

[80] Zhou, Yin Sun, Pei Zhang, Yu Anguelov, Dragomir Gao, Jiyang Ouyang, Tom Guo, James Ngiam, Jiquan Vasudevan, Vijay. (2019). End-to-End Multi-View Fusion for 3D Object Detection in LiDAR Point Clouds.