Integration of driving physical properties into the development of a virtual test field for highly automated vehicle systems

René Degen, M.Sc.

(Cologne University of Applied Sciences, Germany
Uppsala University, Sweden);

Harry Ott, M.Sc.

(Cologne University of Applied Sciences, Germany
Uppsala University, Sweden);

Fabian Overath, M.Sc.

(Cologne University of Applied Sciences, Germany);

Florian Klein.

(Hoersch und Hennrich Architekten GbR, Germany);

Dr. -Ing. Christian Schyr.

(AVL Deutschland GmbH, Germany);

Prof. Dr. Eng. Mats Leijon

(Uppsala University, Sweden);

Prof. Dr. rer. nat. Margot Ruschitzka

(Cologne University of Applied Sciences, Germany);

Abstract

For many years now, models for representing reality have played a decisive role in the development of control systems. By appropriate abstraction they help to design an efficient development process. Especially in the development of Advanced Driver Assistance Systems (ADAS) a valid virtual development environment is crucial for functionality and reliability.

This study aims the representation of driving physics in a virtual test environment for the development of robust ADAS systems. The overall system consists of a georeferenced virtual traffic environment, a multibody vehicle model and a driver model. The virtual environment includes a detailed 3D

model of an urban city in consideration of specific height coordinates of the environment. The vehicle model is implemented by a simplified two-lane model based on geometric steering correlations. Alternatively, the vehicle kinematics are considered by a five-body dynamic model. This model is combined by a semi-empirical tyre model for realistic modelling of the contact forces and torques between the tyre patch and the road. Finally, sensor models for radar, lidar and camera are added to the vehicle model.

To investigate real urban traffic scenarios an advanced driver model is included, which uses a pure pursuit path tracking algorithm to follow a given target trajectory. To investigate real pedestrian interaction, a real persons behavior is included by motion capturing technologies. Those heterogeneous environments are combined by Co-Simulation to get a real-time connection and finally the entire testbed.

By applying the Co-simulation environment to a typical inner city traffic scenario, the verification of the system functionality is done. The outcome is a safe and efficient virtual city environment, which enables interaction investigations between typical traffic participants and highly automated vehicles. In summary, the paper shows the high potential of virtual Co-simulation environments for progressing automated vehicle functionalities.

1. Introduction

Advanced Driver Assistance Systems/Autonomous Driving (ADAS/AD) are becoming more and more important in the automotive industry. It is expected, that automated vehicles will provide promising advantages in transportation and mobility (BMVI, 2015). In 2019 48 % of all new cars sold in Germany were equipped with a lane keeping assistant, 39 % have an autonomous emergency brake and 38 % were delivered with an adaptive cruise control. (Statista, 2020) In 2020 90% of the German car driver were of the opinion that ADAS increase the vehicle safety. 89% thought assistance systems make driving more pleasant. (Statista, 2020) Although the data refer to the German market, a similar result can be expected internationally. This leads to the expectation that the market for ADAS will continue to grow in the future. Besides the opportunities ADAS offer to the vehicle safety, they also increase the vehicles complexity and the testing effort.

According to (BMVI, 2021) Germany creates the legal framework for automated driving functionalities at public German roads. On 28 July a new law was published, which legalizes level 4 functionalities at defined public roads in regular traffic. (BMVI, 2021)

This publication presents a novel approach to combine a highly realistic virtual urban environment by real traffic participants including motion realistic persons and physical correct models for the vehicle and it´s sensors.

In the following Figure the model-structure is shown.

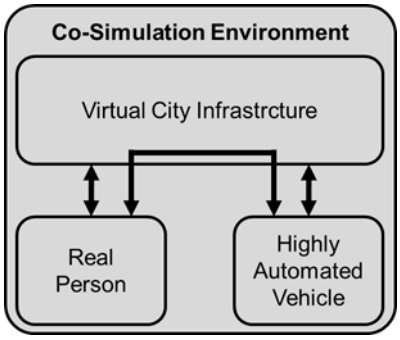

Figure 1: Schematically illustration of the overall model structure

2. Virtual Testbed

The accurate simulation of complex innercity scenarios is an important issue in current automotive research activities. Hence, the following section will introduce an innovative virtual urban traffic environment. The goal is the realistic simulation of complex urban traffic scenarios, considering the interactions of automated vehicles, pedestrians and other road participants. Figure 2 shows the general structure of the used and implemented model network. The core of the research environment is a highly authentic and visually realistic, georeferenced virtual reality city scene. To enable the mentioned interactions, the model is augmented by two sub models. A dynamic pedestrian avatar model, steered by a real-time network motion capturing and a vehicle model, including a physically correct dynamic model and three sensor models. The vehicle avatar serves as an interface to implement real vehicle functions into the scene. Its implementation is done in a closed loop communication between MATLAB and the Virtual Reality engine via network communication. Finally, three sensor models, implemented in the virtual City Model, aim to simulate the recognized surrounding data by the vehicles sensors.

The visualization and potential further processing of the Radar and Lidar datasets takes place in MATLAB. The camera data are evaluated by an Artificial-Intelligent based object detection algorithm. The data transmission is done by network protocol again. The aim is not only to simulate the data in a high realistic way, but also to create a decentralized structure that makes it possible to test complex scenarios independent of the location.

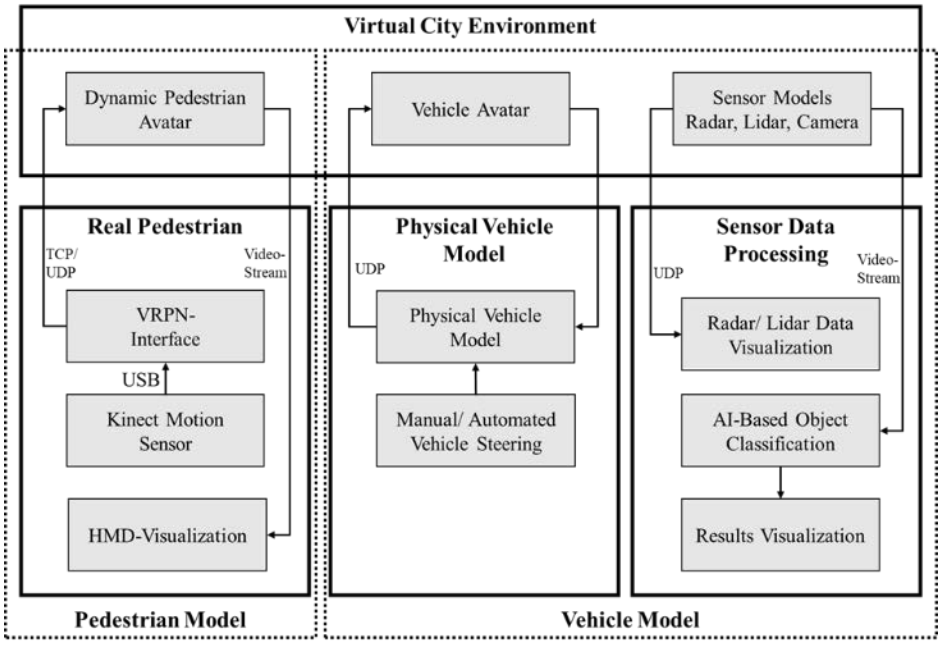

Figure 2: Scheme of the urban traffic model network

a. Virtual City Environment

The first segment of the virtual testbed is the virtual city environment. The environment is not only characterized as the geometric representation of the virtual innercity scene, it also serves as an interface for the model network to be implemented, as shown in Figure 2. Due to that, there are some important requirements for the implementation of the city environment. On the one hand, it needs to be optically as realistic as possible, to enable an authentic simulation on vision based sensors. On the other hand, it needs to be augmented by further metadata for other environmental sensors like radar and lidar. Furthermore, it needs to enable the implementation of avatars for the test vehicle and the real pedestrian subject, including the necessary network interfaces. Since the implementation of the city model is relatively complex, the development environment must be versatile and universally linkable.

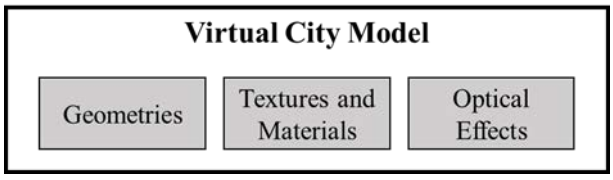

Figure 3: Structure of the virtual city environment

According to Figure 3 the virtual city model can be divided into three sub categories. The geometries provide all necessary physical structures of the buildings, streets and street furniture. Since these models are colourless, the

textures and materials enable a realistic optical representation by applying colours, textures and further surface data to the objects shells. Finally, additional optical effects provide an accurate overall impression for the virtual interurban scene. The geometric virtual city model is based on so-called GIS Data. These are data used by geo information systems, containing rasterized information like infrastructures, land usages, administrative areas, terrain topography, building data or imagery. It is obvious, that not all information provided by GIS datasets are needs. Due to that, the first processing step is the analysis and filtering of these datasets. The result of this first step is a geo referred geometric representation of the city environment. In a next step, these results are reconditioned to be usable within the gaming engine, using third party tools. An example for a processed building with simplified geometries is provided in Figure 4.

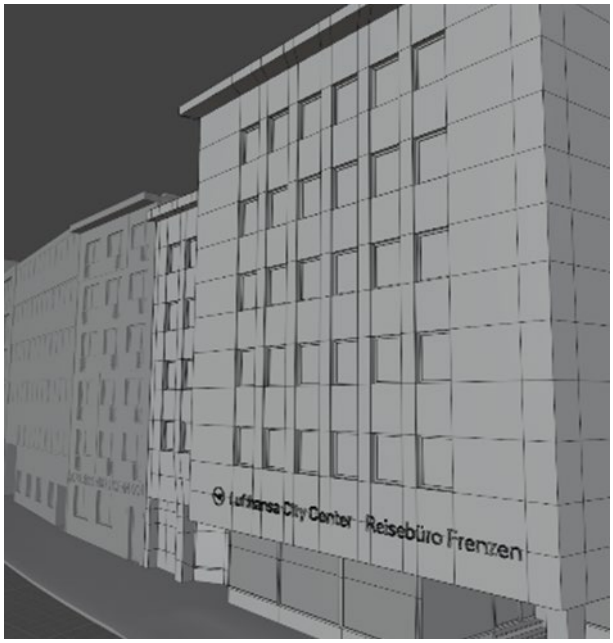

Figure 4: Reconditioned building within 3DSMax

Besides the city itself, the street furniture are an important part in the urban appearance. Objects like postboxes, bins, newspaper vending machines and advertising columns shape the image of the cities. The creation of these objects is done by hand, according to the original objects. Within the Unreal Engine, all street furniture and additional objects are multiplied as often as necessary and placed to fit the appearance of the real environment. The result is an accurate but still uncolored representation of the real environment, as visualized in Figure 5.

Figure 5: Uncoloured city scene

To enable the optically realistic representation of the scenario, the next step is the texturing of the imported geometries. Depending on the respective objects appearance, two general texturing methods are distinguished. For surfaces with simple repeating textures, materials are applied to the whole surface. These are either taken from pre-defined material libraries, or created by hand. For objects with more complex appearances, pictures of the real role model serve as a reference. These images are edited, rectified and then applied to the virtual surfaces.

Figure 6: Comparison of textured objects, real and virtual

An example for that is provided in Figure 6. The left hand side provides a picture of the real object that serves as a reference; the right hand side shows the virtual object for this reference. Hence, it is now possible to visualize the virtual city scene corresponding to the real environment. However, the implementation of the static pedestrian dummies that are used within the scene is different. These dummies are created using a 3D scanning process. This

process directly creates the textures, so that the objects can be imported including the associated appearance.

Finally, optical effects are used to illuminate the virtual environment in a realistic way. Different light sources are used, depending on the lighting situation. For example, at a bright day the sun and the sky light serves as the main light source, at night, the vehicles headlights, the street lights and even illuminated billboards serve as light sources. Additionally, influences like dust, fog or rain can be simulated. This makes it possible to recreate critical lighting situations for the detection of objects with camera based sensors. An example for that is provided in Figure 7. The example represents a situation with low sun and dusty air.

Figure 7: Critical lighting situation with low sun and dust

b. Dynamic Pedestrian Model

As shown in Figure 2, the real human vehicle participant is integrated bidirectional into the virtual testing environment. Its biomechanical behaviour is detected by motion capturing hardware and integrated into the scene in real-time. Additionally, the participant is able to interact with the virtual scene by using a head mounted display. This way a bidirectional interaction is possible.

For the first methodical approach, the Microsoft Kinect for Windows is used. It uses an optical tracking system with an infrared laser emitting a pattern of dots, a monochrome CMOS infrared sensor to capture the dot pattern and an RGB camera to capture the environment. Based on the deformation of the infrared dots in the environment a depth image is created from the environment 30 times per second by the sensor. Next the Kinect focusses on moving objects and evaluates these pixel-by-pixel to identify parts of the human body. (Zhang, 2020) Finally, a skeleton is fitted into the recognized human body, which consists of 20 body joints as shown in Figure 8. Per body joint the position and rotation in quaternions are determined.

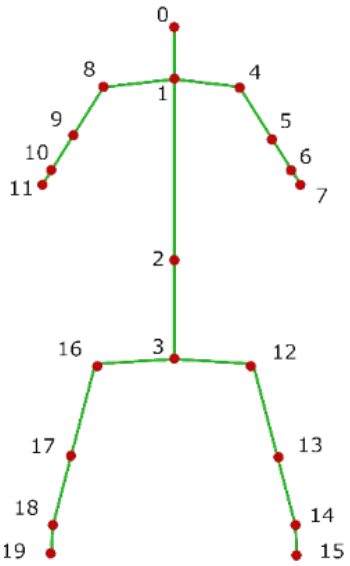

Figure 8: Kinect for Windows skeleton

To forward the data of the Kinect in real-time, the virtual reality peripheral network (VRPN) is used (Taylor, 2020). Therefore, a VRPN Server connects directly with the Kinect via USB. The VRPN standardizes the data received as one person is called a tracker and all 20 body joints are called sensors within this tracker with their respective sensor number and data. The data is then received by the VRPN-Client and directly forwarded further to the Unreal Engine to be used in the dynamic pedestrian avatar.

As motion capturing data is received multiple times a second and data packages are latency critical an unreliable UDP connection is used sending motion capturing data. For establishing a reliable connection and sending status messages between the instances where latency is not that critical, a separate TCP connection is used.

To realize the dynamic pedestrian avatar inside the virtual city environment mesh models with an inherited skeleton are used to display the movements of the tracked pedestrian in real life. Depending on the avatars skeleton, the Kinect skeleton can be directly applied or an additional assignment of the body joints has to be done. In order to keep the proportions of the character independent from the tracked subject only rotations are applied to the avatars skeleton. The position is realized via the hip-centre bone (index 3). Additionally, the avatars feet are automatically calibrated to the ground of the digital environment respecting its topology. As shown in Figure 9, for the first methodical approach and testing, the default Unreal mannequin is used to display the movement of the real pedestrian.

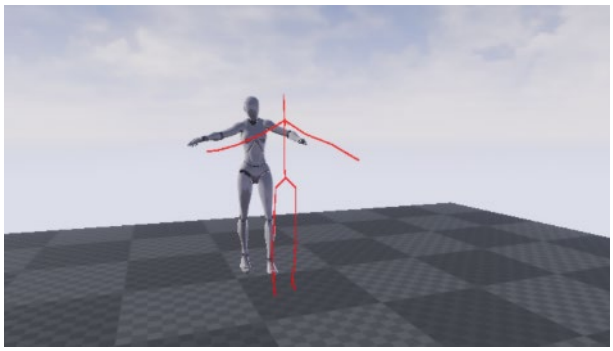

Figure 9: Dynamic pedestrian avatar

c. Vehicle Model

The vehicle model serves as an interface for the implementation of real vehicles and driving functions into the virtual reality. At this point of development, it consists of four sub models. A physical vehicle dynamics and driver model executed in Matlab enables the accurate steering of the artificial vehicle within the virtual reality either manual or automated. Additionally, three sensors models simulate the data of a Lidar, a Radar and a camera based environmental sensor within the virtual scene. These sub models are introduced in the following sections.

i. Physical vehicle dynamics and driver model

To implement a vehicle in the virtual environment accurately, it is necessary to simulate the influences of the vehicle dynamics depending on the scenes environmental influences in a realistic way. Since the model needs to run in real-time, it is important to find a complexity level that represents the cars movements in a sufficient accuracy, without taking to much computation resources. Most Virtual Reality development engines aim not to implement physical simulation models. Therefore, the model is built in MATLAB Simulink. It is composed of different sub models, as shown in Figure 10.

The vehicles vertical dynamics are simulated using a multibody system. Five bodys represent the vehicles main masses, consisting of the bodywork and the four wheels. The stimulation of the system is done at the wheels foot points. This allows the interaction of the model and the virtual city environment, depending on the topography of the scenery. A double track model simulates the lateral dynamics of the vehicle. Additional models represent the drivetrain and the slip angle dependency of the tires as well as environmental influences like rolling, air and gradient resistances.

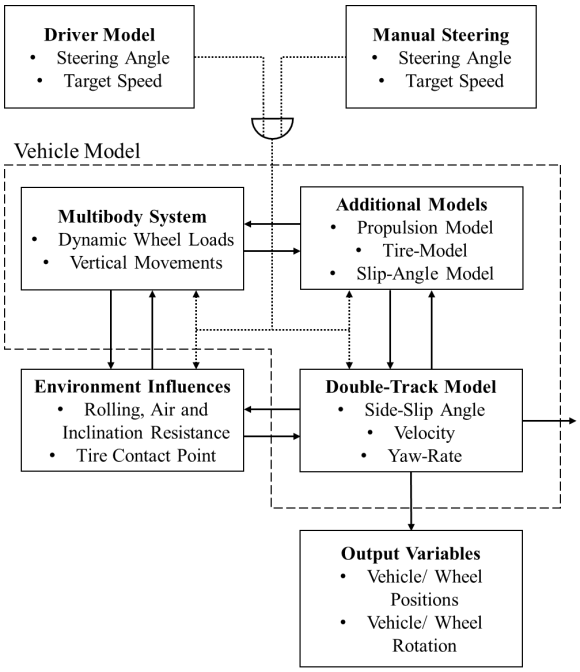

Figure 10: Vehicle model with sub models and steering input

The steering of the model can either be done manually by a user input or automated by a driver model. The automated model uses pre-defined spline paths, created in the virtual city environment and exported to MATLAB. The implemented path following algorithm bases on a pure pursuit controller, like described in (Samuel, 2016). It controls the vehicles steering input by following look ahead points in front of the car lying on the defined spline path.

As already mentioned, the implementation of the vehicle model takes place in MATLAB. Corresponding to Figure 2, a representation of the car in the virtual city environment is needed. Hence, a vehicle avatar is implemented in the Unreal Engine. As the physical model, it consists of five bodies. The UDP protocol is used for communication between the two models. To set the vehicles transformation at the beginning of each computation frame in the Unreal Engine, the positons of the five bodies, received from MATLAB, are used. After that, a so-called line trace is done, to get the height coordinates of the wheel contact patches. These data are frequently sent back to the vehicle model. Thus, a closed control loop is implemented.

ii. 2.3.2. Lidar Model

An environmental sensor that is often used in modern vehicles with ADAS systems is the Lidar. The technology uses laser beams to scan the surrounding. Multiple beams are sent either one after another or at the same time. Since the following model aims to simulate the physical properties of a Lidar, the scanning principle is not relevant for further considerations. It is only important

that a possibility be provided to control the shape and resolution of the Lidar field. For each sent Laser beams the time the light takes to travel to the target object and back to the sensor is determined. According to (Winner et al, 2016). This can be used to compute the distance as described in Equation 1.

$$d = \frac{c_0 \cdot t_{of}}{2} \quad (1)$$

In this equation, c_0 represents the speed of light, t_{of} is the duration the light travels and d is the distance to the target object. Since it is not possible to simulate the speed of light in a virtual environment, a substitution model is needed. For that the so called linetraces or raytraces offer an opportunity to model the laser beams. If a ray hits an object within the scene, the impact location and additional data are returned in a structure. To enable the coverage of the field in front of a car corresponding to a real sensor, the field in front of the virtual car is scanned at every simulation step. The azimuth and the elevation angle define the area of interest. For the discretization of the field, the angular resolution of the sensor in the respective direction is used. With that, the complete azimuth range gets scanned, as shown in Figure 11. Then the elevation angle is incremented and the azimuth angle is scanned again, until the whole field is covered.

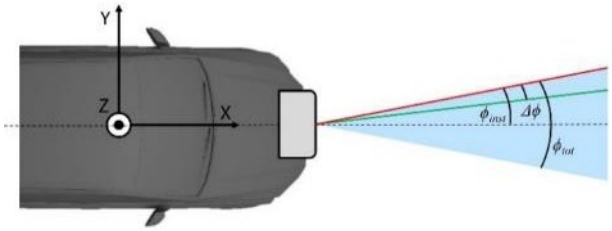

Figure 11: Discreet scanning of the Lidar field

If a ray hits an object, an algorithm is executed, computing the relevant data of the Lidar recognition. The aim is to determine whether a point is detected by the Lidar. An important value for this decision is the signal to noise ratio (*SNR)*, as defined in Equation 2.

$$SNR = \frac{P_r}{P_n} \quad (2)$$

Here P_r represents the power received from the Lidar and P_n is the sum of the induced noise powers. The higher the SNR value, the greater the probability of detection. Corresponding to (Winner *et al.*, 2016), (Kim *et al.*, 2013) and (Kernhof *et al.*, 2018) and through additional adaptations and assumptions the fraction of the received power can be expressed by Equation 3.

$$P_r = \frac{\rho_t \cdot A_r \cdot \tau^2 \cdot P_t \cdot \eta_{sys} \cdot \cos(\theta_i)}{Q_V \cdot \pi \cdot d^3} \quad (3)$$

In this equation, ρ_t is the reflectance coefficient of the target object; A_r represents the receiving lens area; τ stands for the atmospheric transmission coefficient; η_{sys} are the summarized system losses; θ_i is the incidence angle of the light beam on the objects surface and Q_V stands for the divergence of the shot beam. For the implementation of this equation into the virtual reality, most of the parameters can be passed as variables. Only the incidence angle and the targets distance are dependent on the linetraces. The distance can directly be read out of the linetrace results, the incidence angle can be computed by equation 4 using the incidence vector $\underline{i}$ and the surface normal $\underline{n}$ at the impact point.

$$\theta_i = \cos^{-1}\left(\frac{\underline{i} \circ \underline{n}}{|\underline{i}| \cdot |\underline{n}|}\right) \quad (4)$$

Besides the received power, the noise powers acting on the Lidars receiving systems need to be determined. They are mainly composed of the sun induced noise and the dark current noise. The sun induced noise is generated by sunlight illuminating the targets surface and impinging the sensor. Equation 5, according to (Kim et al., 2013), can compute the power of this noise source.

$$P_{sun} = E_{Si} \cdot B_\lambda \cdot \rho_t \cdot A_r \cdot \tau \cdot IFOV^2 \cdot \eta_{sys} \quad (5)$$

E_{Si} represents the illumination intensity of the sunlight, B_λ is the electromagnetic bandwidth of the receiving unit and IFOV is the instantaneous field of view.
Thermal effects of the photo element generate the dark current noise. For the computations of this noise, Equation 6 is used.

$$P_{DK} = \frac{I_D}{\Re_{\max}} \quad (6)$$

Here, I_D stands for the dark current and $\Re_{max}$ represents the maximum sensitivity of the photo element. Both parameters can usually be found in the datasheets of photo elements. With that, it is now possible to compute the signal to noise ratio and make a decision whether a point gets recognized or not. Hence, on every simulations step a point cloud with metadata like the SNR-value and the determined powers is generated. However, it is problematic, that the coordinates of captured points are shown as perfectly accurate values. Real Lidar sensors have a limited resolution due to the time capturing system, amplifications and analog to digital conversions (Kernhof *et al.*, 2018). Since commercial sensors differ in capturing technologies and used hardware, it is not feasible to model these inadequacies accurately. Instead, the influence of the acting inadequacies are displayed. In the given case, this is

done by multiplying the resolution, provided by most sensor manufacturer, with a white Gaussian noise and adding the result to the distance value.

Therefore, all relevant data are known and ready for further processing. To enable this, the values are stored in arrays and sent to MATLAB by an UDP communication. To generate a practical reference, the sensor is parametrized according to the *Valeo SCALA 3D Laser Scanner*, a commercial serial product. The determined values are mainly taken from the sensors datasheet (Hexagon, 2021). All missing values are adopted from the datasheet of a typical photodiode (Hamamatsu, 2018) and the Literatures (Weber, 2018) and (Kim, et al., 2013). The environmental parameters like the atmospheric transmission coefficient or the irradiance of the sun are set dynamically within the virtual scene, depending on the particular study.

iii. 2.3.3. Probabilistic Radar Model

Radar sensors are commonly used in applications for autonomous driver assistance Systems, since they are comparatively cheap, provide a large detection distance and are resistant against environmental influences. The literature provides different approaches for the modelling of Radar sensors. The virtual testbed aims to simulate all necessary environmental data for the interaction of autonomous vehicles and driving functions with pedestrians in real time. Hence, a probabilistic radar model is used. It provides all relevant data in an object list and simulates the phenomena of the radar technology, without performing a full physical simulation.

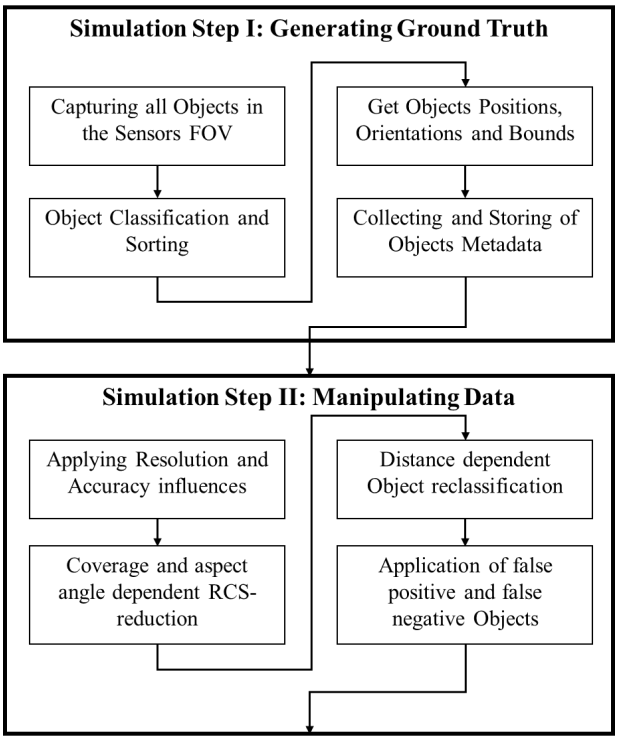

Figure 12: Overview of the probabilistic radar model execution

The implemented model is largely based on the probabilistic radar model presented by (Muckenhuber, et al., 2013) and is extended by several assumptions. To represent the model as realistic as possible, the real commercially used radar sensor Continental ARS 408 serves as a reference. (Liebske, 2015) provides the Datasheet. The execution of the model takes place in two steps, as shown in Figure 12. In a first step, the so-called Ground Truth Data need to be generated. This is a dataset, containing all possibly detectable Objects in the sensors field of view and the corresponding metadata without any errors or inaccuracies. After the perfect data are generated, the second execution step manipulates the dataset with respect to measurement errors, resolutions and inaccuracies. Hence, the phenomena of the sensor are mapped.

For the generation of the Ground Truth all relevant objects lying in the detection area of the sensor need to be captured. The datasheet of the sensor provides the sensing areas displayed in Figure 13.

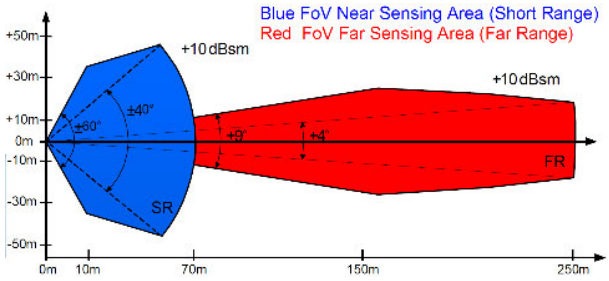

Figure 13: Sensing areas of the Continental ARS 408 (Liebske, 2015)

Since the current research illustrates the interaction of traffic participants in inner-city environments, only the displayed near sensing area needs to be modeled. As with the previous described Lidar model, Linetraces will be used to get the relevant objects, similar to Figure 11. However, at this point, it is not important to get the data of the trace itself; instead, the objects are passed to an array, if they are hit the first time. The angular resolution of the Linetracing needs to be much higher than at the Lidar model, not to omit any object. Moreover, the detection range of the Sensor is not constant over the azimuth angle. Hence, a case distinction is implemented. In the azimuth range of +/- 40°, the tracing length is set to a distance of 70 m. For the range between 40° and 60° a simple approach based on linear equations is used to get the tracing length depending on the azimuth angle. Since not all objects are relevant for the virtual sensor, the next step is the classification and sorting of the objects. Table 1 provides the types of the objects for the classification.

Table 1: Object classifications for the probabilistic radar model

Typ	Index	Color
Car	1	Cyan
Truck	2	Blue
Pedestrian	3	Red
Motorcycle	4	Yellow
Bicycle	5	Green
Unknown	6	Magenta

Based on that, all relevant actors within the virtual scene are augmented by a so called tag. These tags are detected, if the actor is hit. Additionally, a color is assigned to each class for visualization purposes. The positions and the orientations are provided in global coordinates and thus need to be converted into the local sensors coordinate system. Besides the positions and orientations, the bounds or bounding boxes of the recognized actors are relevant for further processings. If an actor only consist of one geometry, like a pillar, or a bin, the bounds can directly be read out. If the actor contains more geometries, like a car, consisting of a body and tires, or a bicycle, the bounds need to be computed.
Finally, only the ideal maximum radar cross sections (RCS) are missing for the recognized objects. This value gives an indication of how large the proportion of the reflected radiation energy is, that impinges on an object. In further processing steps the RCS value can be used to make a decision, if an object is recognized. As the objects index, the value is predefined in tags for each relevant actor within the scene. Since the value fluctuates depending on the aspect angle, the pre-defined value provides the maximum possible radar cross section. Previous works like (Degen, et al., 2021) show the general possibility to simulate radar cross section within a virtual reality engine. However, the paper also offers issues in real-time performance. Due to that, the current probabilistic model uses a pre-defined RCS for every object and manipulates it to get a realistic value.

If a probabilistic model is to be created from the data, the executions visualized in step II of Figure 12 need to be done. The physical phenomena of the sensor are replicated, without simulating its full physics. At first, the influences of the sensors resolution and accuracy are applied to the ground truth signal as displayed in Figure 14.

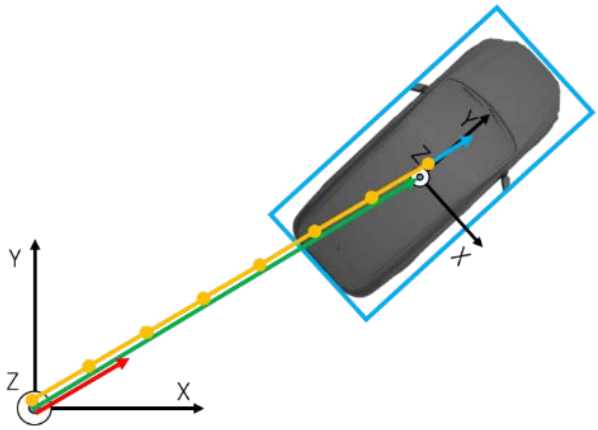

Figure 14: Visualization of the data manipulation to simulate the sensors resolution and accuracy.

The vehicle coordinate system and a theoretical target object are shown. The green vector represents the ideal and error-free position of the target. To implement the resolution influences so called range gates are formed, corresponding to the sensors resolution. To sort the distance into the range gates, the length of the vector is divided by the resolution and rounded to an integer value. This results in the number of range gates. Subsequently the resulting integer value is multiplied with the resolution again. This leads to the yellow vector. The effect of the measurement accuracy is implemented next. This value fluctuates randomly in positive and negative direction. Thus, the value of the measurement accuracy, taken from the datasheet (Liebske, 2015), is multiplied with a white Gaussian noise with a standard deviation of one. The resulting vector, visualized in blue, is added to the yellow in range gates sorted vector.

After the range manipulation, the next step is the implementation of aspect angle and coverage effects to the RCS value. The method for that is visualized in Figure 15. The manipulation is based on the assumption, that the maximum RCS of an object is given at its longest side. In Figure 15 this is represented by the yellow straight. To simulate coverage and aspect angle influences, the recognized length of the target is computed. The result is the straight visualized in red. In a next step, this straight is projected onto the Y-axis of the local sensor coordinate system. To artificially reduce the ground truth RCS, it gets multiplied with the quotient of the length of the yellow straight, representing the objects longest side and the length of the purple straight, representing the visible fraction of the object. This completes the reduction of the RCS.

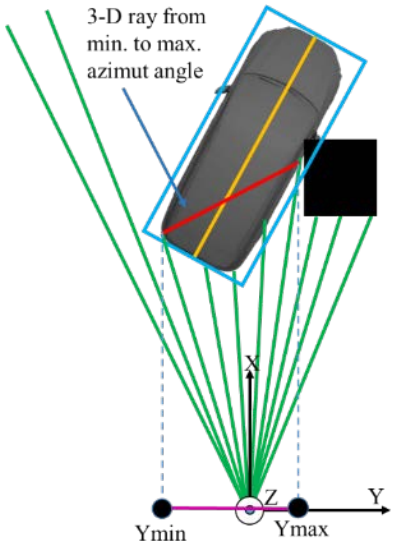

Figure 15: Method for the manipulation of the RCS value

According to Figure 12 the next step of the probabilistic model is the object reclassification. Real radar sensors classify objects on their radar signature. This is only possible in certain distances, depending on the objects type. So far, all objects are recognized and classified correctly. To simulate the real radars detection behavior, two thresholds are implemented. It is assumed, that cars, trucks and motorcycles are correctly classified up to a distance of 50 m. For Pedestrians and bicycles the assumed threshold is 30 m. If the distance of any object exceeds the threshold defined for its class, then it gets reclassified to the “unknown” class.

The last manipulation step visualized in Figure 12 is the implementation of false positive (FP) and false negative (FN) objects. This means objects can appear that are not physically part of the real surrounding (FP) and objects that are physically part of the sensing area (FN) can stay undetected for a certain time. Both phenomena are implemented in the following, starting with the FN objects. A detection probability with a value between zero and one is pre-defined for every Object class. On every execution step and for every actor a pseudo random value is generated. If the generated value is smaller than the detection probability, the object is added to the object list. If it is larger, the object is neglected. With this, it is possible to adjust the average false negative rate for every object class. The implementation of false positive objects is more complex, as the objects are not only to be implemented, but also random positions have to be found for them. At a first step, a value for the average number of recognized FP objects at every frame is defined depending on the object classes. Additionally, typically bounding box sizes are defined. The FP objects are added after the execution of the complete probabilistic radar model. The generation of the FP objects is done for every object class separately,

through a loop that iterates the object types. In that loop the class individual average number of FP objects is used to compute the actual number of FP objects for the respective simulation step, by applying a white Gaussian noise to the value and converting it to an integer. After that, a second loop is initiated for the generated integer value, where the respective FP object is generated. The first value, that is generated for each FP object is the extend. For that, the pre-defined average size of the class dependent bounding box is manipulated by a Gaussian noise. The same procedure is also used to generate an artificial RCS value for the respective object. After all metadata are created, the respective object is positioned at a random position with a random orientation within the sensors field of view. This is repeated until the computed number of false positive objects for the object class is reached. After that, the object class is incremented and the algorithm is executed again.

iv. 2.3.3. Camera Model

The last environmental sensor model implemented in the virtual urban environment is a camera model. Since the camera itself only provides video data, it is augmented by an exemplary application. The resulting video is used in a detection algorithm. The structure of the implemented model, including the detection algorithm and necessary interfaces is visualized in Figure 16

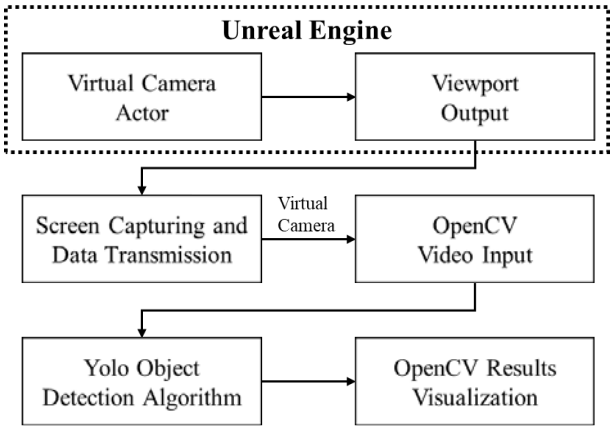

Figure 16: Structure of the implemented camera model, including an object detection algorithm

To generate video data in a realistic way, the used Unreal Engine provides pre-defined camera actors. These cameras enable the simulation of different camera parameters and inadequacies. These camera actors provide the opportunity to simulate almost any camera in a realistic way. Figure 17 shows an example for that. For the generation of the upper image a cheap webcam with many inadequacies is used. The lower image shows the attempt to simulate these inadequacies within the Unreal Engine using a virtual camera actor. It turns out that it is not possible to create an optically identical image; however, it is possible to lower the content of information to the same level.

Since especially the information that can be generated from an image are important for ADAS applications, the virtual camera actor of the Unreal Engine provides a valid method for the generation of image data in further considerations.

The output of the generated video data is done in the so-called viewport and thus directly onto the screen of the user. The resolution is depending on the resolution of the viewport. If the resolution of a physical existing monitor does not fit the demands, it is also possible to use virtual screens. To enable the object detection based on the video output, it is necessary to capture the screen. The interface for the next processing step is a virtual camera. This virtual camera is used as an input by the open computer vision library (OpenCV) in Python. The Python script separates the single frames of the captured video and passes them to the object detection algorithm.

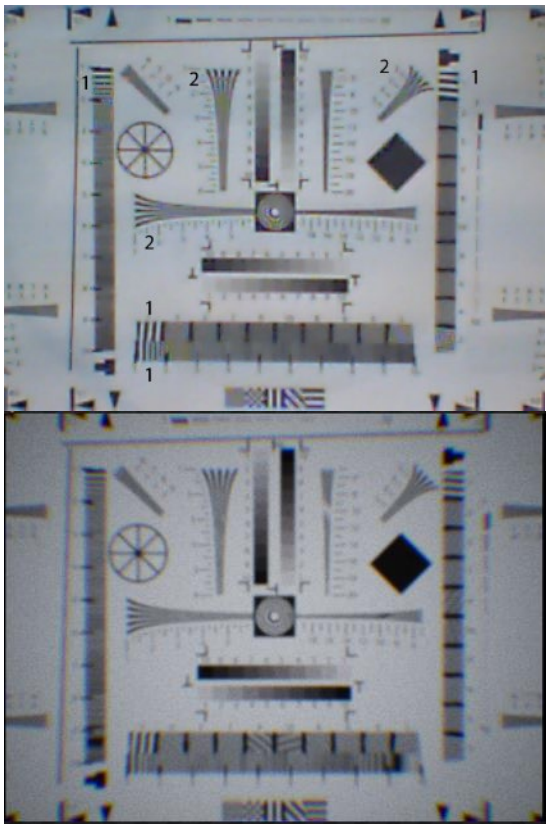

Figure 17: Comparison of real (top) and artificial (buttom) image errors

The algorithm used is the so-called Yolo algorithm that was developed by (Redmon, *et al.*, 2016). Yolo stands for “You only look once” and describes how it works. It recognizes all objects in the respective frame, by looking at the picture only one time. Other algorithms process the picture multiple times. (Redmon, *et al.*, 2016)

As mentioned above, the object classification is implemented in a Python script via OpenCV. Since the Algorithm is based on a neural network, training data are needed. In this study, it is not feasible to train the network with individual datasets. Hence, pre-trained files are used. (Alexey, 2020) provides these in the "model zoo". This model enables the detection of 80 different objects, like bicycles, cars, motorcycle, persons or even objects like animals, fruit, bags and cups. It is not specially designed for traffic scenarios, but contains all relevant object classes. After the execution, the outputs of the model are the object classes, their bounding boxes and the position in the respective image. To visualize the data, OpenCV is used again. The video stream serves as an input and is augmented by the bounding boxes of the recognized objects, labeled with the objects types. Furthermore, the positions, the sizes and the object classes are written into a text file and stored for each analyzed image.

3. Virtual Test Field Verification

The topic of the following chapter is the verification of the previously described and implemented virtual inner-city test field. This is done by testing the implemented vehicle model, including the sensor simulations, and the dynamic pedestrian avatar in a typical urban setup, including further urban furniture and other static traffic participants. The functionality of the virtual test field is assessed based on the general functionality and efficiency of the model as well as the output of the virtual sensors. The aim is to evaluate inadequacies and strengths of the test field.

a. Test Setup

To verify the functionality of all sensor models, it is necessary to implement test objects for all relevant object classes that are described in Table 1, except for the "unknown" class. The used scenario is shown in Figure 18.

Figure 18: Testscene for the virtual testbed assesment

The vehicle marked in red is the vehicle avatar that enables the interaction of the vehicle model with the virtual scene. It also carries the sensor models. Next

to it, a blue car represents the objects class for “cars”. It is to be expected that the vehicle is not completely visible.

Figure 19: Testobject for the "Car" class

The vehicle avatar is standing in front of a pedestrian crossing with traffic lights. Different traffic participants are crossing the road. Figure 20 represents the two pedestrian dummies crossing the road. The left robotic-looking object is the dynamic pedestrian avatar, steered by a test person in real-time. The right object is a static pedestrian avatar that is not able to change its pose. Both objects are assigned to the “Pedestrian class”.

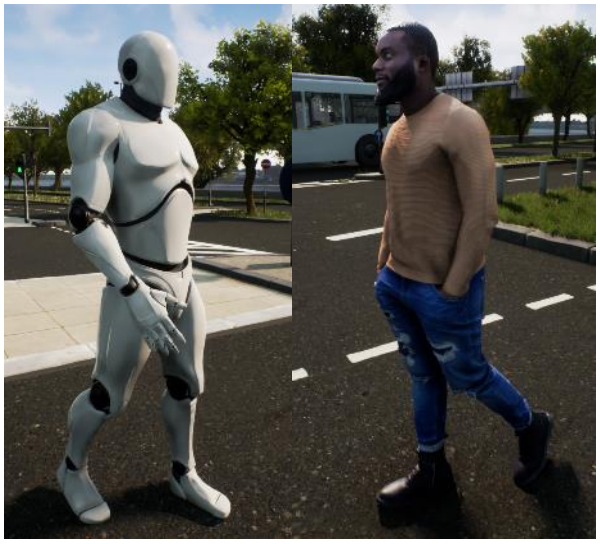

Figure 20: Pedestrian Objects, left dynamic, right static

In addition to the pedestrians, a cyclist crosses the road. The corresponsing model is visualized in Figure 21. It consists of two sub models, the bicycle itself and the rider. The bike is assigned to the bicycle class; the rider is defined as pedestrian. It needs to be investigated, how the sensor outputs are affected by this.

Figure 21: Test object for the bicycle class

A four-lane road runs behind the pedestrian crossing. To represents the motorcycle class, a woman riding a scooter is placed on the closer track of that road. As with the bicyclist, the model consists of two submodels. The scooter is tagged as “Motcycle”, since there is no own class for scooters. The rider is assigned to the “Pedestrian” class. This model is visualized in Figure 22.

Figure 22: Test object for the motorcycle class

The last test object, placed on the more distant lane, is a typical inner-city public bus. The object is assigned to the “Truck” class, since there is no own class for busses. It is visualized in Figure 23.

Figure 23: Test object for the truck class

b. Test Execution

As mentioned in the beginning, the aim of this work is to describe the development and implementation of the urban test field. This chapter serves to test the basic functionality of the models. This is done by a static test in the previously described scenario. Further studies, based on the results of this work, could investigate complex and highly dynamic inner-city scenarios. In order to test the models functionality, a simple static structure is sufficient.

After the scenario is set up corresponding to the last chapter and all relevant metadata are defined for the test objects, the test execution can be done. Since the scenario is static, it is sufficient to capture one simulation step and analyze the results. As described in the respective chapter, the radar and the lidar models are parametrized corresponding to its real equivalents. For the radar model the generation of false positive and false negative objects is activated. The camera is parametrized to a sensor size of 23.76 mm x 13.365 mm, with an aperture of 2.8 and a focal length of 12 mm. This corresponds to a field of view of approximately 90°. The results of all sensor models are displayed on the users screen. For the representation of the radar and lidar data, Matlab is used. For showing the camera model outputs, an Open-CV video stream augmented by the recognized object is shown to the user. To get the corresponding results for a specific time step, the model is executed stepwise, so that it can be paused and the results for a specific frame can be captured. The results are listed below.

c. Test Results

The results of the previously described test are shown in the following, corresponding to the respective sensor.

i. Results of the Lidar Model

The results of the Lidar Model are shown in Figure 24. The scale of the plot is limited to a lateral distance of 40 m and a longitudinal distance of +/- 20 m, so that the region of interest lays within the captured field. All received points, with a Signal-to-Noise ratio smaller than five are neglected. It is assumed, that

this value does not enable a correct recognition. The colored boxes visualized in the plot are added manually, for further explanations. All relevant objects within the scene are recognizable. The points in the purple box near to the test vehicle are generated by the car. The yellow boxes represent the both pedestrians. It shows that the static dummy is recognized as well as the dynamic object. The Lidar plot also detects the step-range of the pedestrian. On closer inspection, the plot shows the front leg and the back leg position of the step. Next to that, the points in the red box are generated by the cyclist and the corresponding bike. The points on the left side, lying in the gray box are reflected by trees and signs. The blue box represents the women on the scooter. It can be seen, that the density of the points is lower, caused by the raising distance. The contour of the bus is also recognizable. It is represented by the green rectangle.

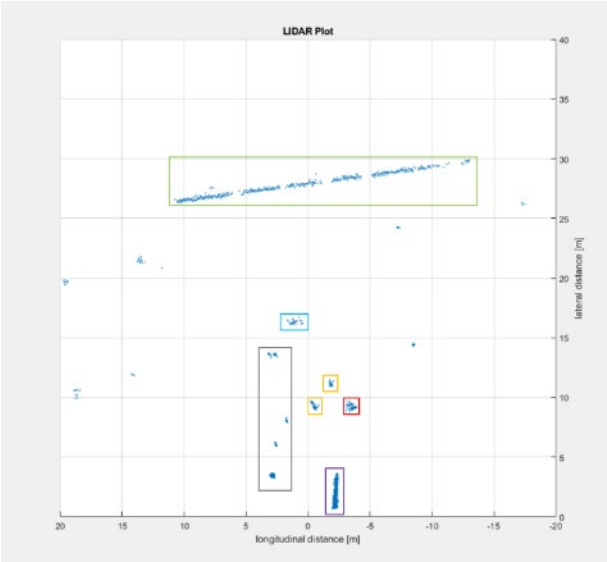

Figure 24: Resulting sensor data of the lidar model

The lidar model shows a valid behavior. All results are plausible and the execution of the model works fluently. Further research and a validation more in depth based on the measured data of real lidar sensors could improve the quality of the model. However, the current implementation shows an adequate possibility for the real-time generation of Lidar raw data.

ii. Results of the Probabilistic Radar Model

The results of the probabilistic radar model are shown in Figure 25. The plot is scaled equivalent to the lidar plot.

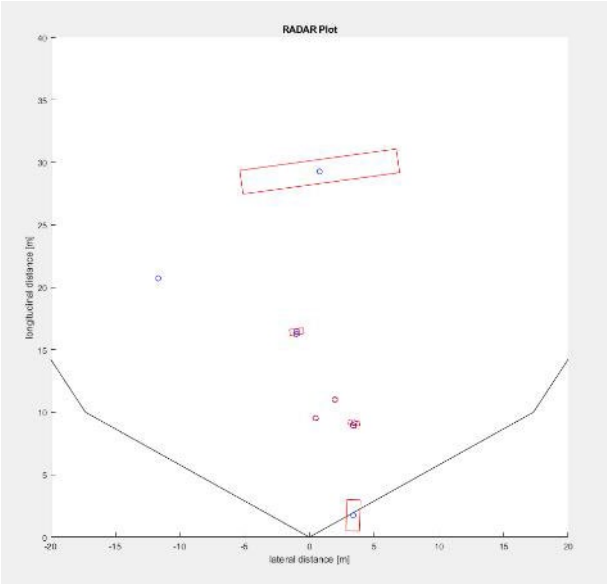

Figure 25: Resulting sensor data of the radar model

The data also show plausible results. All relevant objects lying in the scene that have been tagged are recognized. This shows that no false negative object was generated within the frame under investigation. However, a false positive object can be found at a distance of 20 m on the left side. The object is from the “unknown” class. It can also be seen, that neither the cyclist nor the rider of the scooter is centrally placed on the respective vehicle. This is caused by the implemented inaccuracy of the sensor and the fact, that the riders and the vehicles are treated as separate objects. At this point further investigations could help to find out if it is better to treat the riders and the vehicles as one object or leave them separated. Nonetheless, the results of the sensor are plausible and the performance of the model execution is sufficient. Here a comparison with real sensor data of the described radar system in a real environment could also help to improve the model quality.

iii. Results of the Camera Model

The results of the camera model are shown in Figure 26. The upper image shows the raw camera output of the Unreal Engine that is used as an input for the object recognition algorithm. The lower image show the same augmented by the output of the algorithm.

Figure 26: Results of the AI-based camera model, without and with recognized objects

It shows, that all relevant objects implemented for testing are recognized. The pedestrians, both static and dynamic and the cyclist are recognized as “person” with a certainty of 97 % or higher, whereas the driver of the scooter is recognized with a certainty of 57 %. The bicycle has a certainty of 96 % and the scooter of 73 % to be a motorbike. Additionally, all traffic lights get recognized. The bus lying in the background also is detected, but only with a certainty of 25 %. It is conspicuous, that not only the car next to vehicle avatars is detected, but also the avatar itself, only by a fraction of its bonnet. Lastly, one false detection is done by the algorithm. A backpack that is not part of the scene is recognized. Nevertheless, the sensor model and algorithm proved to be a good example of camera-based object recognition. Further investigations could analyze the influence of different environmental parameters like fog, rain, snow or dirty optics on the detection behavior. It would also be interesting to examine the influence on camera inadequacies like chromatic aberrations or blur on the results.

4. CONCLUSION

The present work shows a novel holistic approach for the accurate testing of autonomous driver assistance systems. It provides a new methodology for the linking of real pedestrians, vehicles and their functions in a highly realistic urban environment. Hence, inaccuracies in real testing procedures, caused by accident hazards, static test environments or just to less test executions are circumvented.

The interaction of the vehicle and its environment is enabled by three different sensor models. These respective models provide an example for the commonly used camera, Lidar and Radar surrounding sensors. Not only the typical data of these sensors are provided, but also typical phenomena and shortcomings are implemented. Furthermore, an accurate vehicle dynamics simulation is used to map the movements of a real vehicle in the virtual environment. To interact with the virtual vehicle directly, this approach enables test subjects to dive into the scenery by a motion capturing and a head mounted display. The person sees the virtual surrounding and their movements are presented by an avatar in the city scene. With that, the model provides a new opportunity for the situational real-time testing of autonomous vehicles and their functions. Due to the already large amount of content, it is not possible to do a complete validation of the implemented models within this work. However, a simple verification approach shows plausible results and confirms the real-time capability of the model. Further researches could analyze the implemented contents in depth, by investigating various scenarios in the virtual environment as well as on real testing grounds. Additionally, it would be interesting to test the reaction of real vehicles and their functions on the virtually provided data.

Acknowledgement

The Project is funded by the Ministry of Economic Affairs, Innovation, Digitization and Energy of North Rhine-Westphalia in the Leitmarktwettbewerb IKT.NRW program.

References

Bundesministerium für Verkehr und digitale Infrastruktur (BMVI) (2015), “Strategie automatisiertes und vernetztes Fahren: Leitanbieter bleiben, Leitmarkt werden, Regelbetrieb einleiten,”

Bundesministerium für Verkehr und digitale Infrastruktur (BMVI) (2021), „Gesetz zum autonomen Fahren tritt in Kraft“ https://www.bmvi.de/SharedDocs/DE/Artikel/DG/gesetz-zum-autonomen-fahren.html

Statista (2020) https://de.statista.com/statistik/daten/studie/1083873/umfrage/anteil-der-pkw-mit-fahrassistenzsystemen-in-deutschland/

Statista (2020). https://de.statista.com/statistik/daten/studie/1108736/umfrage/meinungsumfrage-zu-assistenzsystemen-in-autos-in-deutschland/

Alexey, GitHub, Inc. (2020). https://github.com/AlexeyAB/darknet/wiki/YOLOv4-model-zoo

Degen, R., Ott, H., Overath, F., Schyr, Ch., Leijon, M., Ruschitzka, M. (2021) *Methodical approach to the development of a Radar Sensor model for the Detection of Urban Traffic Participants Using a Virtual Reality Engine. Journal of Transportation Technologies.* 11, 02, 179-195.

Hamamatsu Photonics K.K. (2018). https://www.hamamatsu.com/resources/pdf/ssd/s12023-02_etc_kapd1007e.pdf

Hexagon Autonomy and Positioning (2021). https://autonomoustuff.com/products/valeo-scala

Kernhof, J., Leuckfeld,J. and Tavano, G. (2018). *LiDAR-Sensorsystem für automatisiertes und autonomes Fahren*, in: Thille, T. *Automobil-Sensorik 2*. Springer Vieweg. Berlin, Heidelberg.

Kim, S., Lee, I.and Kwon, Y. J.(2013). *Simulation of a Geiger-Mode Imaging LADAR System for Performance Assessment*. Sensors 13 ,7, 8460-8489

Liebske, R. (2015), Contionental Automotive. ARS 408-21 Premium Long Range Radar Sensor 77 GHz.

Muckenhuber, S., Holzer, H., Rübsam, J., Stettinger, G. *Object-based sensor model for virtual testing of ADAS/AD functions. 2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*

Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2016). *You Only Look Once:Unified, Real-Time Object Detection. IEEE Conference on Computer Vision and Pattern Recognition.*

Samuel, M., Hussein, M. and Mohamad, M. B. (2016). *A Review of some Pure-Pursuit based Path Tracking Techniques for Control of Autonomous Vehicle. International Journal of Computer Applications* 135, 1, 35-38

Taylor, R. (2020). https://github.com/vrpn/vrpn

Weber, H., (2018). *Funktionsweise und Varianten von Lidar-Sensoren*. Sick AG.

Winner, H., Hakuli, S., Lotz, F. and Singer, C. (2016). *Handbook of Driver Assistance Systems*. 1st edn. Springer International Publishing. Cham.

Zhang, Z. (2012). *Microsoft Kinect Sensor and Its Effect. IEEE MultiMedia,* vol. 19, no. 2, pp. 4-12, doi: 10.1109/MMUL.2012.24