Environment Modelling for Robotics and Automated Driving

Project Group Summer Term 2023

submitted to

Institute of Control Theory and Systems Engineering

Faculty of Electrical Engineering and Information Technology

Technische Universität Dortmund

by

Tushar Isamaliya, Elidjana Baka, Christoph Langlitz, Abhishek Marathe,

Jash Sachdev, Shakib Neman, Aman Kungwani, Thibaut Groll

Date of Submission: September 27, 2023

Responsible Professor:
Univ.-Prof. Dr.-Ing. Prof. h.c. Dr. h.c. Torsten Bertram

Academic Supervisors:
M. Sc. Nathalie Müller
M. Sc. Patrick Palmer
M. Sc. Heiko Renz

Nomenclature

Abbreviations and Acronyms

3D	3 Dimensional
AP	Average Precision
BEV	Bird's Eye View
CNN	Convolutional Neural Network
FOV	Field of View
FPN	Feature Pyramid Network
Hz	Hertz
i.e.	id est
IoU	Intersection over Union
LiDAR	Light detection and ranging
m	Meter
mAP	Mean Average Precision
NBV	Next best view
PBR	Physically Based Rendering
ROS	Robot operating system
Sr. no.	Serial number
UR	Universal Robots
VoD	View of Delft
w.r.t	with respect to
YOLO	You only look once

Mathematical symbols

$\mathbf{L}(n)$	Log odds representation

$\mathbf{P}(n)$	Probability of occupancy
f_{oct}	Frequency of Octomap creation
r_{oct}	Resolution of Octomap
s_r	Sensor range of Octomap
t	Time
vf_s	Pcl filter voxel size
x, y, z	Spatial coordinates
z_t	Sensor measurement at time t

Abstract

The first chapter is based on Radar, Lidar and Camera fusion for environment perception in autonomous vehicle navigation. In the rapidly evolving field of autonomous vehicles, the challenge for precision and robustness in perception systems has led to the integration of diverse sensor modalities. The synergy between multiple sensors not only enhances accuracy but also introduces complex challenges, with sensor fusion being a pivotal concern. Recent years have witnessed a surge in research focused on 2D and 3D object detection, semantic segmentation, and object tracking, all driven by the need to harness the complementary strengths of various sensors. Sensor fusion, aims to combine the strengths of different sensors for enhanced accuracy. One area of focus is the fusion of LiDAR and camera data, known as deep fusion, and radar and camera data, termed centerfusion. The fusion methodologies discussed in this context exhibit significant promise within the field of 3D object detection, particularly within the scope of autonomous driving applications. Deep fusion optimally elaborates on mitigating the gap between LiDAR and camera features when fusing them, thereby enhancing the ability of accurate object detection. Centerfusion delves into the utilization of radar, characterized by its robustness and capacity for long-range object detection. However, it is important to acknowledge that each of these fusion strategies presents distinct innovations and their presence is significant in the ongoing research and development of autonomous vehicle navigation.

The second chapter presents a comprehensive study on environment modeling for automated driving scenarios within Unreal Engine 5, focusing on adverse weather conditions. Leveraging scenes from the Adverse Weather Dataset, we explore the critical task of simulating realistic weather scenarios and assessing their impact on automated driving systems. The core objective of this research is to evaluate the performance of Unreal Engine 5 by carrying out YOLOv5-based vehicle and pedestrian detection algorithms. We begin by developing a high-fidelity environments within Unreal Engine 5, similar to the scenes from the original dataset. Following the scene setup, we implement the YOLOv5 object detection model. The YOLOv5 model is fine-tuned and tested in the simulated adverse weather scenarios, enabling the evaluation of its performance in detecting vehicles and pedestrians under challenging conditions. The essence of the package lies in the comparative analysis between the detection results obtained in the scenes from the original Adverse Weather Dataset and those generated within Unreal Engine 5. Scenes undergo manual optimization to align the confidence scores as closely as possible with the original detections. In summary, it is observed that Unreal Engine 5 offers the robust capabilities necessary for crafting scenes tailored to Automated Driving simulations and is the most cost efficient solution to the devised problem.

The third chapter focuses on the development of an environment modelling system for the Universal Robots UR10 robotic platform utilizing the Octomap framework in the Robot

Operating System (ROS) environment. The primary objective was to create a comprehensive and accurate three-dimensional representation of the robot's surroundings, enabling it to perceive and navigate within complex, dynamic environments. To achieve this, we integrated the UR10 robot with Microsoft Kinect Azure camera which has depth camera, allowing for the collection of dense point cloud data. Through extensive experimentation and evaluation, the system showed improved metrics to capture the details of the environment as volumetric representation. Additionally, we modified the package to switch between dynamic and static obstacles. The static Octomap excels in providing realistic object modelling. Notably, making changes in resolution and applying voxel filtering has demonstrated improvements in its performance. One of the Octomap's strengths lies in its dependable modelling, especially when adjusting the resolution, particularly with lower resolutions. This feature contributes to safety as lower-resolution Octomaps are designed to overlap higher-resolution Octomaps of the same object. This ensures that object representations do not shrink in size when working with lower resolutions. The outcomes of this project bear significant implications for a wide array of applications, including object recognition and human-robot collaboration. The ability of the UR10 robot to construct and continuously update a spatial representation of its surroundings lays the foundation for enhanced awareness, ultimately advancing its capabilities in real-world, unstructured environments

Contents

1 Introduction

To make systems work autonomously in today's world, proper modelling of the environment of is very important in order to function accurately, precisely and optimally. The automated system such as Automated driving and Industrial robots should have proper information about their environment to assess the situation and interact accordingly. There are various method to model the environment and a way to select the method is by considering the system, the Hardware/Software combinations and the type of surrounding. The aim of this project was to evaluate different methods of environment modelling for Automated driving and for Robotics. Throughout this report, we explore the algorithms and methodologies used in the project.It is further divided into chapters categorized according to the system and method used to model the environment.

The first chapter is based on Radar, Lidar and Camera fusion for Environment Perception in Automated Driving. In the rapidly evolving field of autonomous vehicles, the quest for precision and robustness in perception systems has led to the integration of diverse sensor modalities. The synergy between multiple sensors not only enhances accuracy but also introduces complex challenges, with sensor fusion being a pivotal concern. Recent years have witnessed a surge in research focused on 2D and 3D object detection, semantic segmentation, and object tracking, all driven by the need to harness the complementary strengths of various sensors. Sensor fusion, a pivotal concern in this quest, aims to combine the strengths of different sensors for enhanced accuracy. One area of focus is the fusion of LiDAR and camera data, known as deep fusion, and radar and camera data, termed centerfusion. These fusion approaches hold great potential for 3D object detection in applications like autonomous driving. While deep fusion leverages the complementary nature of LiDAR and cameras, centerfusion taps into radar's resilience and long-range capabilities. However, each approach has unique challenges and advantages, making them vital subjects of exploration in the development of autonomous vehicles.

The second chapter is based on Environment Modelling for Automated Driving in Unreal Engine 5.1. Recreation of 20 specific and static scenarios from the Adverse Weather and Lighting Conditions Dataset is done in the Unreal Engine 5.1. Parameter optimization with the help of modular architecture which consists of adjusting material instances, light, camera, post process parameters, etc. Data exchange between Unreal Engine and MATLAB to analyze pictures in MATLAB and Python. After this, evaluation of taken pictures in the YOLOv5 model is done to get the confidence and bounding box prediction. Manual optimization of various parameters in recreated scenes till the prediction of vehicle and pedestrian detection matches the prediction with the Adverse Weather Dataset conditions.

The third chapter is based on three-dimensional mapping and occupancy modelling of the object using the UR10 robot and Microsoft Azure Kinect camera. The ROS Octomap package was implemented which subscribes Point cloud messages published by Depth camera and build Octomap to map and model the Object. Octomap's ability to represent the environment as a probabilistic volumetric map offers a valuable foundation for object recognition, collision avoidance, and path planning. The primary objective of this experiment was to evaluate and optimize the modelling of the environment and surrounding object using Octomap mapping. The task involves capturing different viewpoints to map unknown voxels of the object using a UR10 robot.

2 Radar and Camera Fusion for Environment Perception

2.1 Overview

One prominent area of exploration in sensor fusion revolves around combining LiDAR and camera technologies for 3D object detection. Cameras are very commonly used in automotive applications, as they provide high resolution information. However, it is important to note that it is a 2D sensor. It is highly dependable on technology to translate the captured data into 3D interpretations. LiDAR on the other hand, provides precise 3D measurement data. Lidar, as in 'Light Detection and Ranging', makes use of laser light beams to measure distances and give highly accurate maps of the area, remotely. These maps not only detect and position objects but also identify what they are, and whether they are static objects or in the move. In fields like autonomous vehicle navigation, where the main objective, the one of safe and effective operation, requires accurate, thorough and detailed maps of the environment. This is why, for these applications, providing accurate depth and reflection intensity, LiDAR makes for an important supplementary to the Camera Vision to achieve accurate and robust perception. However, this fusion approach is not without limitations. Both LiDAR and cameras are susceptible to adverse weather conditions, such as fog, rain, or snow, which can impair their effectiveness. Moreover, neither LiDAR nor cameras inherently provide information about object velocity, a crucial element for collision avoidance in many scenarios, potentially hindering their use in time-critical situations (Nabati and Qi 2020). Radar sensors have resilience in adverse weather conditions, they are capable to detect objects at impressive long ranges (up to 200 meters in automotive applications), and have the ability to estimate object velocities using the Doppler effect. Despite these inherent advantages, radar has remained relatively unexplored in sensor fusion research. A key obstacle in this domain is the scarcity of datasets containing radar data for autonomous driving applications, which poses challenges for researchers (Nabati and Qi 2020). Radar point clouds are notably sparser, rendering them unsuitable for precise extraction of objects' geometric information. While aggregating multiple radar sweeps can increase point cloud density, it introduces delays into the system. Furthermore, the vertical measurements such as elevation of radar points are often inaccurate or missing, as automotive radars typically report only distance and azimuth angles to objects (Nabati and Qi 2020). This work delves into the underexplored terrain of fusing radar data with camera sensors, highlighting on the unique strengths and challenges of radar technology in the context of autonomous driving.

2.1.1 Innovative Approaches to Sensor Fusion in 3D Object Detection

The integration of multiple sensing modalities has become a cornerstone in enhancing the accuracy and robustness of perception systems. This report explores two sensor fusion strategies that harness neural networks that combine data from diverse sensor modalities such as camera, radar and LiDAR. To optimize the fusion of data from different sensors, a fusion of schemes has been employed within neural networks. One approach, known as early fusion, combines raw or pre-processed data from various sensor modalities are fused together. With this method, the network learns a unified representation from the different sensing sources. However, early fusion methods can be sensitive to spatial or temporal misalignments in the data. Conversely, late fusion occurs at the decision level which is at where the outputs or information from multiple sensors or modalities are aggregated or combined to arrive at a final decision or conclusion. It provides greater flexibility for integrating new sensing modalities into the network but may not fully exploit the potential of the available sensors, as it does not leverage the intermediate features acquired from learning a joint representation. A middle ground between early and late fusion approaches, termed middle fusion, extracts features individually from different modalities and combines them at an intermediate stage. This approach strikes a balance between sensitivity and flexibility (Simonelli et al. 2019).

2.1.2 Lidar- Camera Sensor Fusion

In this section we focus on that part of the perception process, the fusion between a LiDAR and a camera. LiDAR - Camera fusion enables accurate position and orientation estimation but the level of fusion in the network matters. Several approaches have been introduced on position estimation, and all existing works focus on vehicles. Our work is based on (Kim et al. 2023). Camera images also provide considerable success in 3D object detection tasks, and numerous Deep Neural Network architectures have been developed for LiDAR-based 3D object detection. These detectors encode LiDAR point clouds using backbone networks and detect objects based on the features obtained from the encoder. Another widely used LiDAR encoding method is Voxel-based encoding. This method voxelizes LiDAR points in 3D space and encodes the points in each useful information for 3D object detection (Deng et al. 2021). Visual features obtained by applying convolutional neural network (CNN) model to camera images can be used to perform 3D object detection as well (Deng et al. 2021). To leverage diverse information provided by two multiple sensors, a feature-level fusion strategy has been proposed, which uses both voxel features and camera features together to perform 3D object detection. These methods involve the challenge that two features are represented in different coordinate domains (i.e., camera-view versus voxel domains), so one of the coordinate representations must be aligned and adapted into another without losing information of the original domains. The purpose is to develop a method to reduce such a large domain gap in dual-domain features, in order to improve the performance of camera-LiDAR sensor fusion methods, in terms of coordinates and data distribution when fusing their features. In Figure 2.1, it is explained the concept of this approach. The 3D Dual-Fusion approach employs a Dual Fusion Transformer (DFT) to enable dual-domain interactive feature fusion (Kim et al. 2023). The DFT first performs a 3D local self-attention to encode the voxel-domain queries. Then it applies a dual-query cross attention. The latter performs simultaneous feature fusion in both domains, camera-view and voxel through Dual-queries. Dual-query deformable attention layer uses two types of queries, hence dual-query, to refine features in both domains through multiple layers interactively. Using this strategy, it is possible to maintain and preserve information that might get lost if the interactions between subsequent domains are not modeled. Prior to feeding the lidar features to the Dual Fusion Transformer, an adaptive gated fusion network (AGFN) is used to enhance the camera-domain image features. It projects the voxel- domain Lidar features into camera-domain and then it combines them with the camera-domain features. This feature fusion produces bird eye's view feature maps, which can then be fed into any 3D object detection head, in order to generate a 3D bounding box and classification score. The proposed modules are added to the VoxelR-CNN framework. We conducted experiments and validated the performance of this approach on the Kitti dataset, followed by the implementation and evaluation of the approach on the more recent View-of-Delft.

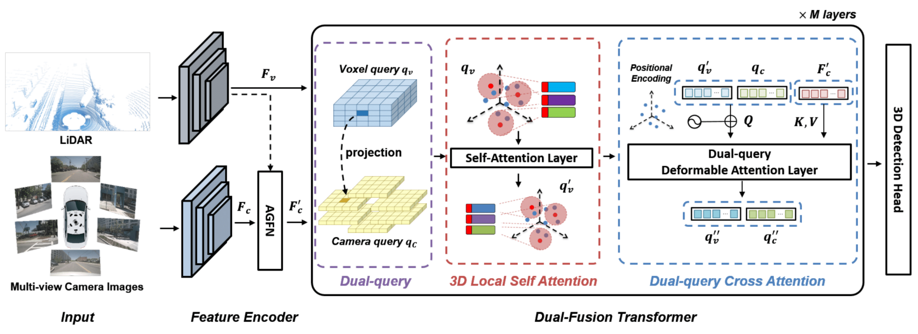


Figure 2.1: Architecture of the 3D Dual Query Camera-LiDAR Fusion

2.1.3 Radar and Camera Sensor Fusion

In this report regarding CenterFusion, a novel middle-fusion methodology designed to leverage radar and camera data for 3D object detection. CenterFusion's core focus lies in associating radar detections with preliminary detection outcomes obtained from images. It accomplishes this by generating radar feature maps and integrating them with image-based features to achieve highly accurate 3D bounding box estimations for objects. The process begins with the generation of preliminary 3D detections using a key point detection network. A novel frustum-based radar association technique is then proposed to precisely link radar detections to their corresponding objects in the 3D space. These radar detections are subsequently mapped onto the image plane, where they contribute to the creation of feature maps that complement the image-derived features. The fused features collectively enable the accurate estimation of objects' 3D attributes, including depth, rotation, and velocity. (Nabati and Qi 2020) The network architecture for CenterFusion is illustrated in Figure 2.2.

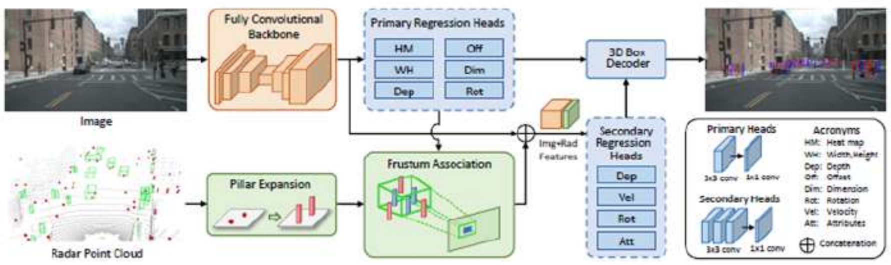


Figure 2.2: CenterFusion Network Architecture

Within the CenterFusion network architecture in Figure 2.2, the initial step involves acquiring preliminary 3D bounding boxes. These preliminary boxes are derived from the image features extracted by the backbone network. Subsequently, the frustum association module comes into play, leveraging the preliminary boxes to establish associations between radar detections and objects. This process also yields radar feature maps, which are instrumental in enriching the dataset. The network then proceeds by concatenating the image and radar feature maps. This merged feature set plays a crucial role in refining the preliminary detection results. Specifically, it recalculates critical parameters such as depth and rotation, essential for precise 3D object localization. Moreover, this combined feature set facilitates the estimation of objects' velocity and other pertinent attributes, contributing to a comprehensive and accurate perception of the surrounding environment.

2.2 Implementation

In the following section we will describe the implementation of the two approaches, 3D Dual-Query Camera-LiDAR Fusion and CenterFusion. We will start with the implementation of 3D Dual-Query Camera-LiDAR Fusion on the KITTI dataset and View-of-Delft. Then we will continue with the implementation of CenterFusion on the NuScenes and the View-of-Delft dataset.

2.2.1 Implementing 3D Dual-Query Camera-LiDAR Fusion on the KITTI dataset and View-of-Delft

Regarding the implementation on the KITTI dataset, but even the general configuration, the proposed modules for 3D Dual-Fusion were added to the backbone network and detection head of Voxel R-CNN (Deng et al. 2021). The project is built on OpenPCDet. The configuration details are as followed: The range of the point cloud data was within [0, -40, -3, 70.4, 40, 1], so [0; 70.4] x [-40; 40] x [-3; 1]m on (x; y; z) axis. The point clouds were voxelized with each voxel size of 0:05 x 0:05 x 0:1m. The maximum number of points per voxel was set to 5 and the number of Dual-Fusion layers, M was set to 4. We used the data augmentation in Voxel R-CNN (Deng et al. 2021). In addition, we used the point-image GT sampling augmentation as per the original instructions of (Kim et al.

2023) and (Zhang, Z. Wang, and Loy 2021). To visualize the BEV map with the bounding boxes with their respective classification scores, we used the Mayavi tool.

The official KITTI evaluation does the evaluation of object detection performance using the PASCAL criteria (Everingham et al. 2010), also used for 2D object detection. PASCAL includes a task called instance segmentation. The main idea of the instance segmentation is that we have to segment out each instance of each category, so that, for example, two vehicles in an image will have the same category label as 'Car', but different instance labels as in 'Car1' and 'Car2'. To evaluate a certain algorithm on instance segmentation, we need the metrics, AP (Average Precision) and mAP(Mean Average Precision). Precision, basically is measuring the percentage of correct positive predictions among all predictions made; and recall is measuring the percentage of correct positive predictions among all positive cases in reality. The Average Precision, it is defined as the area under the precision-recall curve (PR curve), where the x-axis is recall and y-axis is the precision. In other words, precision measures how accurate are the predictions. i.e. the percentage of predictions that are correct and recall measures how good are found all the positives.

AP was calculated using recall 40-point precision (R40) and 11-point precision (R11) (Zhang, Z. Wang, and Loy 2021). The 11-point Interpolated Average Precision was originally proposed in (Salton and McGill 1986), and was used in the PASCAL VOC challenges (Everingham et al. 2010). In Pascal VOC2008, an average for the 11-point interpolated AP is calculated. First, it divides the recall value from 0 to 1.0 into 11 points — 0, 0.1, 0.2, . . . , 0.9 and 1.0. Next, it computes the average of maximum precision values for these 11 recall values. According to the original researcher (Salton and McGill 1986), the intention in interpolating the precision/recall curve in this way is to reduce the impact of the "wiggles" in the precision/recall curve, caused by small variations in the ranking of examples. On (Geiger et al. 2013), they suggest that we use 40 recall positions instead of the 11 recall positions proposed in the original Pascal VOC benchmark. This results in a more fair comparison of the results.

The mean average precision(mAP), is simply all the Average Precision values averaged over different different classes/categories. However, as mentioned above, the official KITTI evaluation of object detection is performed using the PASCAL criteria. It divides the dataset into three difficulties, Easy, Moderate and Hard, by using a few parameters to filter objects and categorize the, into the respective Difficulties. So, we have respective Average Precision (AP) metrics for each class and each difficulty (Geiger et al. 2013). Difficulties are defined as follows:

- Easy: Min. bounding box height: 40 Px, Max. occlusion level: Fully visible, Max. truncation: 15 %.
- Moderate: Min. bounding box height: 25 Px, Max. occlusion level: Partly occluded, Max. truncation: 30 %.
- Hard: Min. bounding box height: 25 Px, Max. occlusion level: Difficult to see, Max. truncation: 50 %.

The Truncation parameter filters the ground truth labels e.g., for the Easy category, it removes objects that are more than 16% (0.16 < truncation) truncated and so on for the three difficulties. Far objects are also filtered based on their bounding box height in the image plane. The occlusion awareness filter's labels w.r.t the objects' occlusion state, whether it is difficult to see, partly visible or fully visible. As in the official KITTI benchmark leaderboard, the results state for the 'Moderate' Difficulty, this category is of our interest.

The following are the used AP metrics calculated for each difficulty: AP_3D for 3D object detection task and mAP metric AP_BEV for BEV object detection task. For cars, requires a 3D bounding box overlap of 70%, while for pedestrians and cyclists we require a 3D bounding box overlap of 50%. The model Voxel_RCNN (Deng et al. 2021) was trained on the dataset Val SPLIT and the results show for a competitive performance leading the current state-of-the-art approaches on the KITTI official leaderboard results (Geiger et al. 2013).

2.2.2 Implementing CenterFusion on the NuScenes and the View-of-Delft dataset

This evaluation of CenterFusion on the nuScenes dataset reveals that it surpasses all prior camera-based object detection methods in the 3D object detection benchmark. (Nabati and Qi 2020) Furthermore, this demonstrate that the incorporation of radar data substantially enhances velocity estimation for objects, all without relying on temporal information. CenterFusion represents a compelling frontier in the field of sensor fusion, wherein it extends the horizons of attainable outcomes within the domain of 3D object detection. Moreover, it enhances the functionalities of autonomous systems, thus contributing to the advancement of this technology.

nuScenes Dataset for Autonomous Driving

The nuScenes dataset is a comprehensive and widely utilized resource in the field of autonomous driving research. It provides an extensive collection of real-world data captured in urban environments, offering valuable insights and challenges for the development of autonomous vehicles. Here is some essential information about the nuScenes dataset.

- Dataset Overview: The nuScenes dataset is designed to support research in various aspects of autonomous driving, including perception, motion forecasting, and planning. It comprises high-definition (HD) sensor data collected from various sensors, such as LiDAR, radar, cameras, and GPS, mounted on autonomous vehicles.

- Sensor Modalities: The dataset includes data from multiple sensor modalities, allowing researchers to explore sensor fusion techniques. It contains data from 32-beam LiDAR sensors, 360-degree camera systems, radars, and accurate GPS/IMU sensors.

CenterFusion: Radar and Camera Fusion for 3D Object Detection

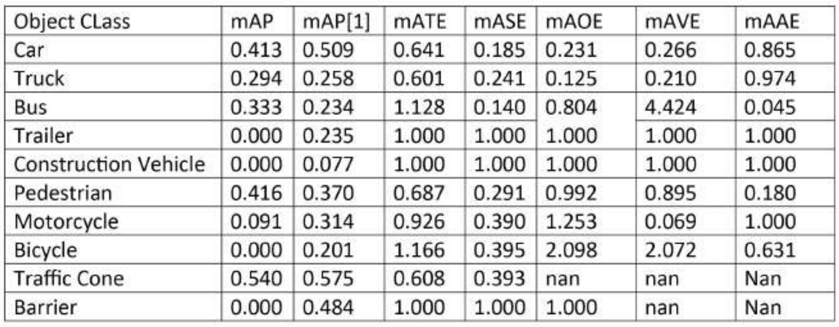
In this section, CenterFusion - for the fusion of radar and camera sensors in 3D object detection is described. The holistic architecture of CenterFusion is depicted in Figure 2.2. To detect objects in the image plane, this leverage the center-based object detection network proposed in (Zhou, Koltun, and Krähenbühl 2020). This network identifies the central points of objects and subsequently predicts various object properties, including 3D location, orientation, and dimensions. This approach hinges on a middle-fusion strategy that accomplishes two critical objectives: associating radar detections with their corresponding object center points and harnessing both radar and image features to enhance preliminary object detections. This enhancement involves recalculating essential attributes, such as depth, velocity, rotation, and object-specific attributes. A pivotal element of this fusion mechanism revolves around achieving precise radar detection-to-object associations. The center point object detection network generates a distinctive heat map for each object category within the image. The peaks within these heat maps pinpoint potential center points for objects. Leveraging the image features at these identified locations, it can estimate various object properties. However, to fully exploit radar data, it is imperative to accurately map radar-based features to the precise centers of their corresponding objects in the Figure 2.4. This mapping necessitates a highly accurate association between radar detections and objects within the observed scene.

Center Point Detection

This approach employs the CenterNet (Zhou, D. Wang, and Krähenbühl 2019a) detection network to generate initial object detections within the image. Initially, image features are extracted using a fully convolutional encoder-decoder backbone network. This report adhere to the CenterNet's methodology by utilizing a modified Deep Layer Aggregation (DLA) network [3] as chosen backbone. These extracted image features serve as the basis for predicting object center points on the image plane, as well as other object attributes such as 2D size (width and height), center offset, 3D dimensions, depth, and rotation. These predictions are facilitated by primary regression heads (see in Figure 2.2), each consisting of a 3×3 convolutional layers with 256 channels and a subsequent 1×1 convolutional layer to produce the desired output. This process yields accurate 2D bounding boxes and preliminary 3D bounding box estimations for all detected objects within the scene.

Radar Association

The center point detection network relies solely on image features at the center of each object to predict all other object properties. To fully leverage radar data in this context, it is crucial to establish an association between radar detections and their corresponding objects on the image plane. One approach might involve mapping each radar detection point to the image plane and associating it with an object if it falls within the 2D bounding box of that object. However, this approach is not robust due to several factors: multiple radar detections may correspond to a single object, radar detections may not relate to any object, and the imprecise or nonexistent z-dimension of radar detections can lead to inaccurate mappings, especially if objects are occluded. Frustum Association Mechanism: To overcome these challenges, this introduce a frustum association method that uses an object's 2D bounding box, estimated depth, and size to create a 3D Region of Interest (RoI) frustum around the object in Figure 2.8. This frustum serves as a filter, significantly reducing the number of radar detections requiring association. Subsequently, the estimated object depth, dimension, and rotation are used to define a RoI surrounding the object, further refining the set of radar detections associated with it. If multiple radar detections exist within this RoI, this select the closest point as the corresponding radar detection for that object. The RoI frustum approach simplifies the association of overlapping objects, as they are separated in 3D space and possess distinct RoI frustums. It also addresses multi-detection association challenges, ensuring that only the closest radar detection within the RoI frustum is linked to the object. However, it does not fully resolve the issue of inaccurate z-dimensions, as radar detections may still fall outside the ROI frustum due to height inaccuracies. Pillar Expansion: To tackle the height inaccuracy problem, this introduce a preprocessing step known as "pillar expansion" for the radar point cloud. This step involves expanding each radar point into a fixed-size pillar (as depicted in Figure 2.4). Pillars provide a more accurate representation of physical objects detected by radar, associating these detections with dimensions in 3D space. With this representation, a radar detection is considered within a frustum if any part of its corresponding pillar falls inside the frustum, as shown in Figure 2.2. This method helps address inaccuracies in height information, enhancing the association process.



Object CLass	mAP	mAP[1]	mATE	mASE	mAOE	mAVE	mAAE
Car	0.413	0.509	0.641	0.185	0.231	0.266	0.865
Truck	0.294	0.258	0.601	0.241	0.125	0.210	0.974
Bus	0.333	0.234	1.128	0.140	0.804	4.424	0.045
Trailer	0.000	0.235	1.000	1.000	1.000	1.000	1.000
Construction Vehicle	0.000	0.077	1.000	1.000	1.000	1.000	1.000
Pedestrian	0.416	0.370	0.687	0.291	0.992	0.895	0.180
Motorcycle	0.091	0.314	0.926	0.390	1.253	0.069	1.000
Bicycle	0.000	0.201	1.166	0.395	2.098	2.072	0.631
Traffic Cone	0.540	0.575	0.608	0.393	nan	nan	Nan
Barrier	0.000	0.484	1.000	1.000	1.000	nan	Nan

Figure 2.3: Evaluation and results on nuScenes

Expanding Radar Points to 3D Pillars (Top Image in Figure 2.4): The initial step involves transforming radar points into 3D pillars (depicted in the top image in Figure 2.4 with red and blue colour). However, directly mapping these pillars to the image and replacing them with radar depth information leads to suboptimal association results. This method results in inadequate alignment with the center of objects and a proliferation of overlapping depth values (as seen in the middle image with green colour). Frustum Association Mechanism (Bottom Image with green colour in Figure 2.4): In contrast, frustum association mechanism offers an effective solution. It accurately maps radar detections to the center of objects while significantly reducing instances of overlapping depth values (as observed in the bottom image). Under this approach, radar detections are exclusively associated with objects possessing valid ground truth or detection boxes, and only if all or part of the radar

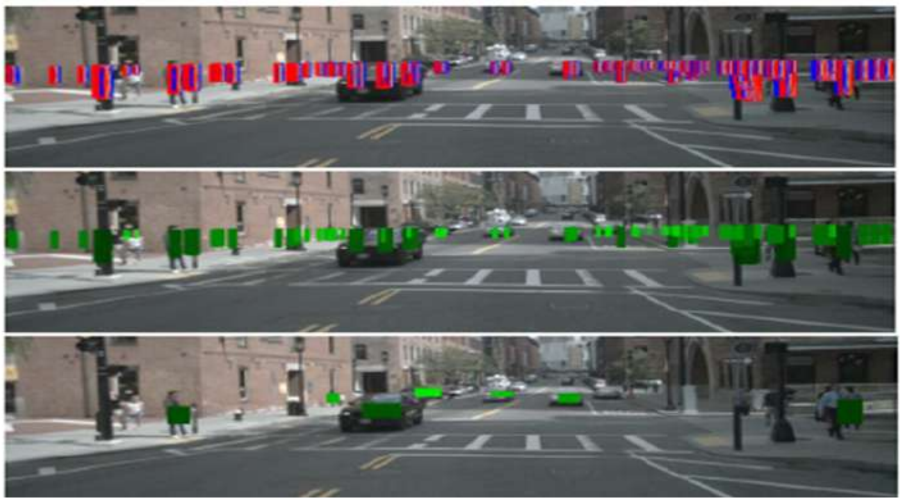


Figure 2.4: 3D Pillars

detection pillar falls within the box. Moreover, the frustum association method prevents the erroneous association of radar detections originating from background objects, such as buildings, with foreground objects. This distinction is clearly illustrated in the case of pedestrians on the right-hand side of the image.

Implementation Details

In this implementation, it employs the pre-trained CenterNet (Zhou, D. Wang, and Krähenbühl 2019a) network, which utilizes the DLA (Yu et al. 2018) backbone as the core object detection network. The DLA network features iterative deep aggregation layers designed to enhance the resolution of feature maps. While CenterNet has explored various backbone architectures, this opt for the DLA network due to its efficient training time and reasonable performance, making it a practical choice. This report leverages the pre-trained CenterNet model, which was trained for 140 epochs on the nuScenes dataset. However, this model does not include predictions for velocity and object attributes by default. To address this limitation, it is train for additional velocity and attribute heads for 30 epochs, ultimately using the resulting model as our baseline. The secondary regression heads integrated into our network are built upon the CenterNet backbone and are trained over an additional 75 epochs, with a batch size of 4 and utilizing Nvidia GPUs.

Challenges with CenterFusion

CenterFusion implementation (Nabati and Qi 2020) heavily relies on a pretrained CenterNet (Zhou, D. Wang, and Krähenbühl 2019b) model for its object detection. Without a suited model CenterFusion will not be able to perform sensor fusion or produce bounding boxes at all.
This creates some dependencies. On the one hand CenterFusion is dependent on the code base and implementation of CenterNet. On the other hand an additional dependency come into play. Just by following the documentation of CenterFusion it is not possible to create a working environment. To overcome that, this project group uses the environment

and implementation of another sensor fusion approach, called CenterFusion++ (Kübel and Brandes 2022). CenterFusion++ receives the resulting model from CenterFusion and trains on it. CenterFusion++ implementation builds on CenterFusions implementation and complements their implementation with their functionality. So, three parties were involved in the coding of the implementation this project group uses to evaluate CenterFusion. Because of lacking documentation, it is not always clear who implemented what or if somebody even removed or changed functionality from the previous code base.
The following emphasizes key aspects of the existing implementation and gives a description on what was found out to make CenterFusion work.

Convertion scripts

In order to train and test CenterFusion models a dataset needs to be handed over to the architecture. Therefore when starting a learning or detection task one need to input a file with the necessary information about the images, the pointcloud and the annotations. This information must be stored in the file in a structured manner, a format that is called coco (Lin et al. 2014). Every dataset needs its own convertion script, which creates the above mentioned file in the coco format.
On goal of this project group work is the evaluation of an image/radar fusion approach on the View-of-Delft Dataset. To acomplish this, a convertion script, designed for the use of the View-of-Delft dataset, has to be implemented. That way the needed file in the coco format gets created.

VoD adaption

The View-of-Delft dataset (Palffy et al. 2022) is a novel automotive dataset recorded in Delft, the Netherlands. It contains over 8600 frames of image, radar and LiDAR data with over 120000 labels including pedestrians, cyclists and cars. The View-of-Delft dataset is currently the largest open-source dataset available.
To create the above mentioned file there are two possible ways. Either with the use of a convertion script that takes the original Vod data structure as an input or to save implementation work with the help of an already existing convertion script that converts a dataset that is similar to VoD to coco. The CenterFusion implementation holds a script that converts the Kittitrack dataset to coco (Geiger et al. 2013). Because the Kittitrack dataset is very similar to VoD it seems to be advantageous to structure the VoD data into the Kittitrack data structure and use a modified convertion script to convert the data into coco.
At first it was verified that the original Kittitrack convertion file works fine. Kittitrack is an open-source dataset that is freely to use. After downloading the dataset and use the convertion script it becomes clear that the convertion file is working and that CenterFusion can train and detect on it. Bounding boxes appear and prove that both, CenterFusion and the convertion script seem to work. It becomes clear that CenterFusion is not only working on nuscenes what it was originally created for. The original task in this work package is it to adapt the approach to the VoD dataset. After organizing the VoD data like the Kittitrack structure some modifications in the original Kittitrack convertion script must be made. First, Kittitrack does not use pointcloud data, images only. So, the script must

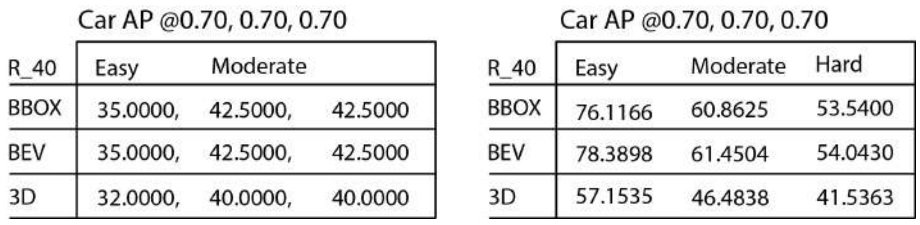
be modified to also write the VoD radar data into the coco formatted file. Additionally, the intrinsic camera matrix and other information about the pixel high and width must be included.
After training and testing the dataset, it shows a similar result like with Kittitrack. CenterFusion displays bounding boxes on the VoD dataset and proves that it is adaptable to other datasets.

2.3 Evaluation

In the following section we will describe the evaluation of the two approaches, 3D Dual-Query Camera-LiDAR Fusion and CenterFusion. We will start with the evaluation of 3D Dual-Query Camera-LiDAR Fusion on the KITTI dataset and View-of-Delft. Then we will continue with the evaluation of CenterFusion on the View-of-Delft dataset.

2.3.1 Evaluating 3D Dual-Query Camera-LiDAR Fusion on the View-of-Delft dataset

As the official evaluation provides an AP metric only on the Hard difficulty, we decided to follow the OpenPCDET (*OpenPCDet* 2023) and KITTI official evaluation. As we previously stated, it is of our interest to observe the performance on the Moderate difficulty. The official KITTI evaluation divides the dataset into three difficulties, Easy, Moderate and Hard. It is important to note that the official KITTI evaluation uses a "Truncation" parameter based on which, filters the ground truth labels e.g., for the "Easy" category, it removes objects that are more than 15% (0.15<truncation) truncated and so on for the three difficulties. On the official website of the View-of-Delft Dataset (Palffy et al. 2022), it is explained that the creators did not annotate the truncation levels. Instead, they used this field for other properties during their experiments, e.g. whether the object is moving (1) or not (0), or as a Track-ID (0,1,2,3,4) using integers. In case of the KITTI dataset a percentage is used.
In our evaluation, any object that was not static but moving, was removed. This was fine for cars, because many of them are static, but for pedestrians and cyclists it was certainly the case. So, we did not use the truncation levels in our evaluation. We also did not filter samples based on the number of points. This case concerns more the fusion with radar sensors, as Radar has much fewer points than LiDAR. But even in our experiments, this step removed most pedestrians and cyclists, and the results could not be negligible.
The model Voxel_RCNN (Deng et al. 2021) was trained on the dataset Val SPLIT. The results of the experiments regarding the different number of decoder layers on the baseline showed an improvement when increasing layers from 2 to 4.
The table above in Figure 2.5 shows the difference in Average Precision with number of decoder layers equal to 2 (on the left) and equal to 4 (on the right). This goes to show that using the decoded queries as input queries for the next Dual-Fusion Transformer layer, makes for a key component in this approach (Kim et al. 2023).
This feature fusion as shown in Figure 2.6 produces bird eye's view feature maps, which



Car AP @0.70, 0.70, 0.70

R_40	Easy	Moderate	
BBOX	35.0000,	42.5000,	42.5000
BEV	35.0000,	42.5000,	42.5000
3D	32.0000,	40.0000,	40.0000

Car AP @0.70, 0.70, 0.70

R_40	Easy	Moderate	Hard
BBOX	76.1166	60.8625	53.5400
BEV	78.3898	61.4504	54.0430
3D	57.1535	46.4838	41.5363

Figure 2.5: Results of the experiments regarding the different number of decoder layers on the baseline

can then be fed into any 3D object detection head, in order to generate a 3D bounding box. The illustration shows the results of the detections, represented in the BEV maps and the image and scene, to which the point cloud corresponds too. We note that there is no detection for the Truck class, as we are only experimenting with the three classes, Car, Pedestrian and Cyclist. At a first look, there seems to be correctly detecting the instances of the three classes. In order to get a better understanding of the results, we turn to the table of object detection tasks.

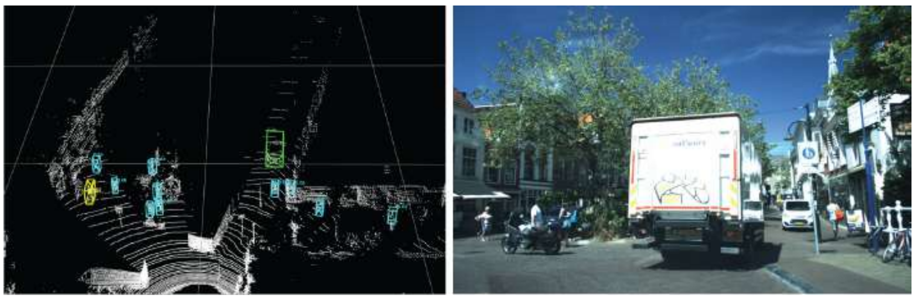
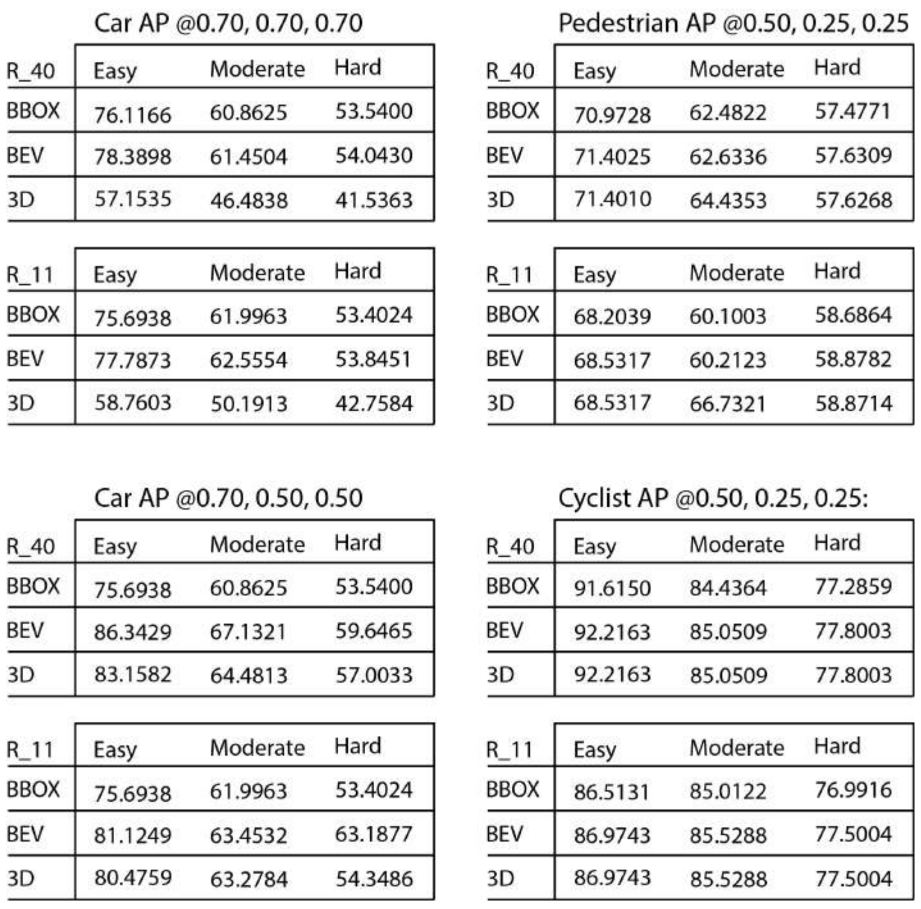


Figure 2.6: Results of the detections, represented in the BEV maps and the corresponding scene

The table in Figure 2.7 shows the average precision metrics on bounding boxes, bird-eye-view and 3d object detection, with different IoU (Intersection over Union) values used for the classes: ’Car’, ’Pedestrian’ and ’Cyclists’. On the Moderate difficulty, which is our interest due to its representation of the partly visible objects, for the Car class, there are over 60% correct positive predictions for the Bird-Eye-View, Bounding Boxes and 3D Object detection task. When the threshold for the BEV and 3D metrics is lowered from 0.70 to 0.50, the performance metric increases.
By not filtering samples by number of points and removing the filtering by “truncation”, we were able to perform a more correct evaluation for Pedestrian and Cyclist classes, as before, most of them were removed and the initial results were very poor. Making this change did not affect the Car class, as in our evaluation, any object that was not static but moving, was removed and the cars are mostly static.



Car AP @0.70, 0.70, 0.70

R_40	Easy	Moderate	Hard
BBOX	76.1166	60.8625	53.5400
BEV	78.3898	61.4504	54.0430
3D	57.1535	46.4838	41.5363

R_11	Easy	Moderate	Hard
BBOX	75.6938	61.9963	53.4024
BEV	77.7873	62.5554	53.8451
3D	58.7603	50.1913	42.7584

Pedestrian AP @0.50, 0.25, 0.25

R_40	Easy	Moderate	Hard
BBOX	70.9728	62.4822	57.4771
BEV	71.4025	62.6336	57.6309
3D	71.4010	64.4353	57.6268

R_11	Easy	Moderate	Hard
BBOX	68.2039	60.1003	58.6864
BEV	68.5317	60.2123	58.8782
3D	68.5317	66.7321	58.8714

Car AP @0.70, 0.50, 0.50

R_40	Easy	Moderate	Hard
BBOX	75.6938	60.8625	53.5400
BEV	86.3429	67.1321	59.6465
3D	83.1582	64.4813	57.0033

R_11	Easy	Moderate	Hard
BBOX	75.6938	61.9963	53.4024
BEV	81.1249	63.4532	63.1877
3D	80.4759	63.2784	54.3486

Cyclist AP @0.50, 0.25, 0.25:

R_40	Easy	Moderate	Hard
BBOX	91.6150	84.4364	77.2859
BEV	92.2163	85.0509	77.8003
3D	92.2163	85.0509	77.8003

R_11	Easy	Moderate	Hard
BBOX	86.5131	85.0122	76.9916
BEV	86.9743	85.5288	77.5004
3D	86.9743	85.5288	77.5004

Figure 2.7: Metrics of 3D object detection, bounding boxes and BEV tasks

2.3.2 Evaluating CenterFusion

The goal of this evaluation is to investigate whether the explained approach of using a convertion script to convert any dataset, in this case VoD, is expedient and is expected to provide reliable bounding boxes. For that, a model will be trained on the VoD-dataset and used to perform a detection task on it.

1. mAP (mean Average Precision):
    - mAP is commonly used to evaluate object detection and image segmentation models.
    - It measures the precision of the model's predictions across multiple classes or categories.
    - Calculated by taking the average of the precision values for each class, where precision is the number of true positive predictions divided by the total number of positive predictions.
2. mATE (mean Average Translation Error):
    - mATE is used to assess the accuracy of a model's translation (e.g., machine translation) by measuring the average error in the position of translated words or phrases.
    - It's often used in natural language processing (NLP) tasks.
    - Calculated by comparing the positions of words in the reference and translated sentences and taking the mean of these errors.
3. mASE (mean Average Scale Error):
    - mASE is a metric used in computer vision to evaluate the accuracy of object scale estimation.
    - It measures the average error in estimating the size or scale of objects in an image.
    - Calculated by comparing the estimated object scale to the ground truth scale for multiple objects and taking the mean of these errors.
4. mAOE (mean Average Orientation Error):
    - mAOE is often used in computer vision tasks like object detection and pose estimation.
    - It assesses the accuracy of orientation or pose estimation by measuring the average error in the predicted orientation compared to the ground truth.
    - Calculated by comparing the estimated orientation to the true orientation for multiple objects and taking the mean of these errors.
5. mAVE (mean Average Velocity Error):
    - mAVE is used to evaluate the accuracy of velocity predictions, often in the context of motion tracking or prediction tasks.

- It measures the average error in estimating the velocity of objects.
- Calculated by comparing the estimated velocities to the true velocities for multiple objects and taking the mean of these errors.

6. mAAE (mean Average Acceleration Error):
    - mAAE is similar to mAVE but assesses the accuracy of acceleration predictions.
    - It measures the average error in estimating the acceleration of objects.
    - Calculated by comparing the estimated accelerations to the true accelerations for multiple objects and taking the mean of these errors.

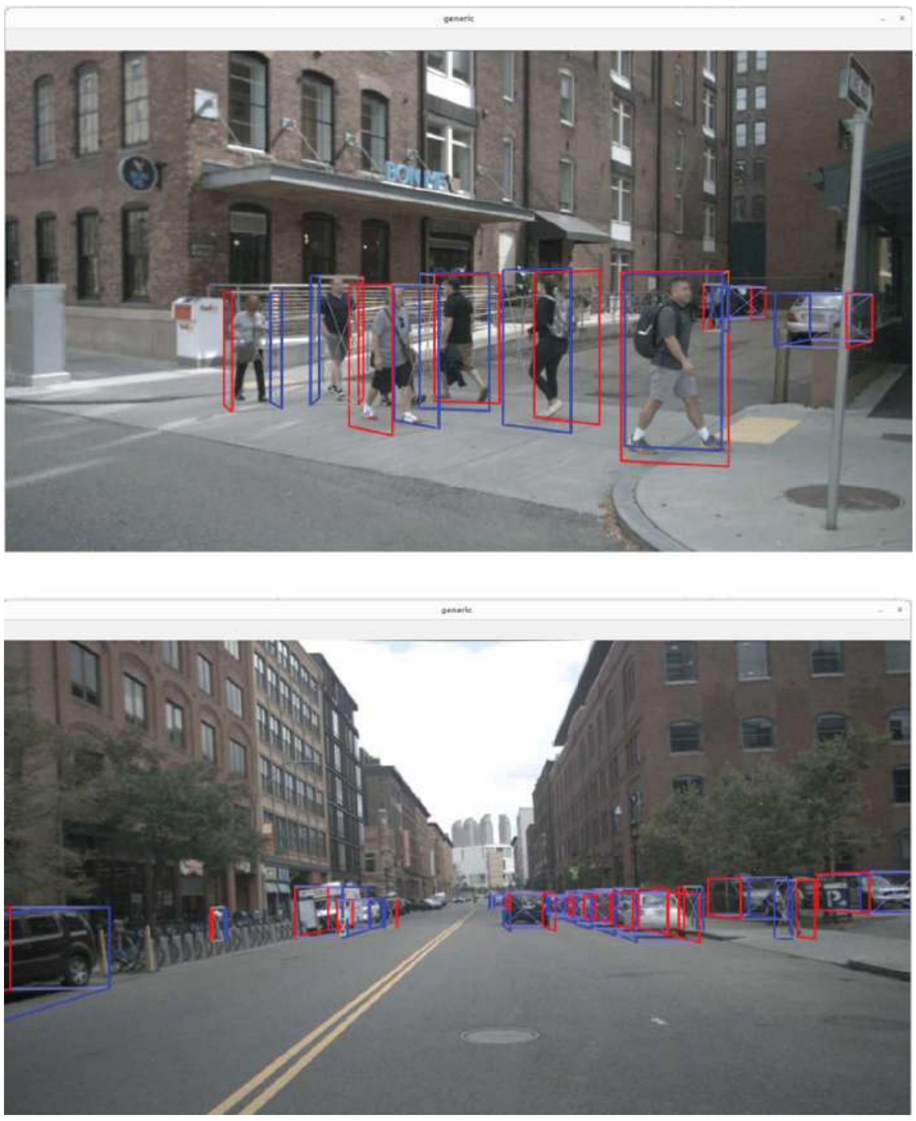


Figure 2.8: Qualitative results of CenterFusion

Overall Performance

In the context of this research, the comparison between the model's mean Average Precision (mAP) and the reference mAP yields noteworthy insights. It is evident that this model's mAP scores exhibit variability across different object classes. Notably, some classes, such as Car and Pedestrian, display improved mAP values compared to the reference, suggesting the model's effectiveness in these scenarios. However, this positive performance is countered by classes like Trailer and Barrier, where the reference mAP surpasses our model significantly. This disparity emphasizes the importance of class-specific performance evaluation and optimization efforts, particularly for classes where the model's mAP falls short. Further investigations and fine-tuning are required to enhance this model's object detection capabilities and achieve consistent high performance across all object classes.

Training

As recommended by the authors of CenterFusion++ the model was trained in two steps. The first step includes training for 50 epoches with a batch size of one. For 50 epoches a model will be trained on top of the pretrained center point detection network while the backbone is freezed. For that, a pretrained model from the CenterNet repository was used. This network was trained for 140 epoches on the nuscenes dataset.
The second step includes training for another 10 epochs and a batch size of one. This time while the backbone is unfreezed to fine tune the model. The resulting model from step one acts as the pretrained model now where we train on top on.

Detection

The detection confidence treshold is set to 0.17, which means that only detections with a confidence score of at least 0.17 will be displayed and evaluated. The dataset is only being evaluated on a subset of the VoD dataset, on approximately 10% of the whole dataset. The following graphics show images and metrics from the VoD dataset while being applied to the self-trained detection model. Graphic 2.9a illustrates that the overall detection works. From a visual evaluation it is obversable that bounding boxes are estimated on the VoD dataset. They seem to be well placed and aligned in a reasonable way.

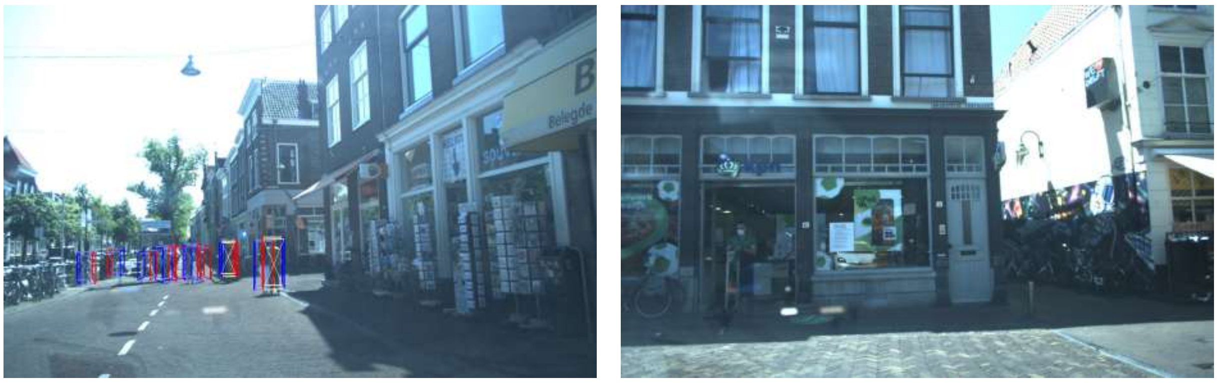

(a) Image 830 (b) Image 73

Figure 2.9: Object detection. Score: 0.17

Not every object is detected on the images and even images like 2.9b that show no bounding boxes at all exist. It seems like the model is not able to reliably detect objects on datasets other than nuscenes as the detection on the Kittitrack dataset also shows similar results like the ones on VoD. A solution to increase the amount of bounding boxes would it be to lower the detection confidence treshold. As a side effect the amount of false positives will increase significally.

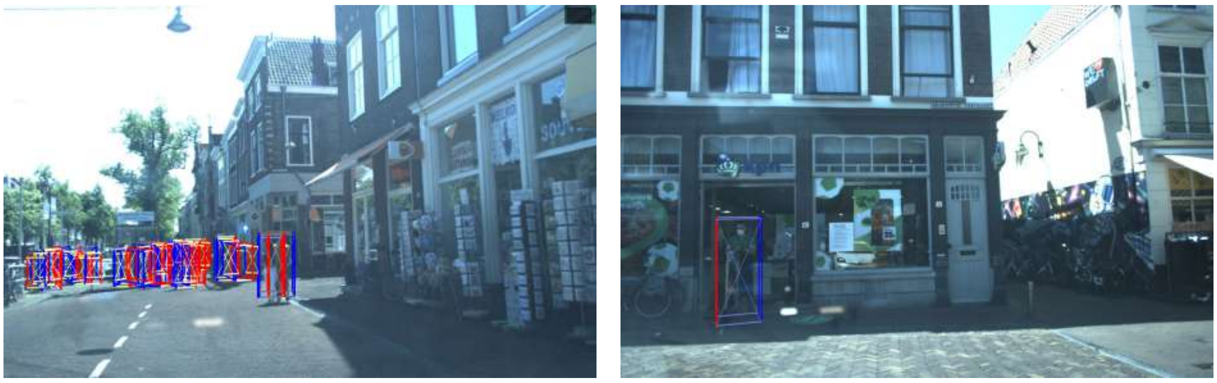

(a) Image 830 (b) Image 73

Figure 2.10: Object detection. Score 0.12

In general the overall confidence throughout the whole dataset and all objects seem low. The average confidence lies at 25% percent.
The table 2.1 presents the previously discussed metrics for CenterFusion on the View-of-Delft dataset.
The maP results in 2.1 seem low. For the bird eye view metrics the mAP is at 36.63% on cars, at 9.83% on pedestrians and at 4.04% on cyclists. The bounding box mAP draws a similar oberservation. The mAP is at 20.6% on cars, at 6.06% on pedestrians and at 1.82 on cyclists. These in general low scores are influenced by the choice of the detection confidence treshold and the pretrained model. The genereal detection confidence of the objects are low and therefore a low treshold was choosen what increases the amount of false positives.

Method[mAP]	**Car** [mAP]	**Pedestrian** [mAP]	**Cyclist** [mAP]	**Overall** [mAP]
3D	20.60	6.06	1.82	9.49
BEV	36.63	9.83	4.04	16.84

Table 2.1: Results of the radar and camera fusion for environment perception

2.3.3 Comparison/Discussion

In this evaluation report, it assessed the performance of the radar and camera fusion network, CenterFusion, on the nuScenes benchmark. Several classes, including "Trailer," "Construction Vehicle," "Bicycle," and "Barrier," pose significant challenges for the model, as evidenced by their low mAP scores. Further investigation is required to understand why these classes are particularly difficult to detect and classify. It's possible that these classes are underrepresented in the training data or have unique characteristics that are not well-handled by the fusion network. Addressing these challenges and implementing potential improvements could lead to better overall performance in future iterations of the model.

Regarding, the training approaches itself for the Radar-Camera fusion as well as for the Lidar-Camera one. The number of epochs may not be enough, as some models need hundreds of epochs to train. However, while experimenting with the training it seems unlikely that more training epochs will significantly improve the detection quality as increasing the epochs did not improve the detection. Additionally, when training on the initial datasets, and Kitti, with the same number of epochs it showed better results, and the majority of objects are detected. We believe this aspect does not affect the performance metrics.
For the CenterFusion, we observe that the problem lies in the in the choice of the used dataset as Kittitrack and VoD show similar results. In comparison to the original CenterFusion approach, only two things were changed by this group. Different datasets were used and an adapted convertion file was implemented. The dataset and the convertion script itself, as well as any configuration issues are unlikely to be the issue, as some objects are detected on Kittitrack and VoD. The overall approach of detecting objects and forming bounding boxes around it seems to be working.
Going back to the introduction of this section, it was explained that CenterFusion uses a pretrained detection model to detect objects solely looking at the image data, as it then continues to concentrate the radar data with it. In other words, this means that the whole sensor fusion approach from CenterFusion is dependent on the object detection in the first place. Without these detections CenterFusion is not able to assign radar data and the whole attempt fails. As indicated several times the pretrained model is very important and it seems to be the limiting factor here. We use the pretrained detection model that is provided by CenterTrack, which was trained on the nuscenes data. This explains why the CenterFusion approach works well with the nuscenes dataset. The pretrained model was trained on the same data and works fine with the sensor fusion in the aftermath. It is expected that a well-trained pretrained model will significantly increase the detection confidences and therefore result to better detections and less false positives.

We stay consistent on emphasizing the key elements of the 3D Dual-Query Dual Fusion regarding LiDAR-Camera Fusion. We observe a significantly better performance of the 3D Dual-Query Dual Fusion approach. The difference in strategy from Centerfusion is that 3D Dual-Query Dual Fusion employs a Transformer (DFT) to enable dual-domain interactive feature fusion. And data preparation Using the Adaptive Gated Fusion Network to enhance the Camera - LiDAR features and using multiple layers to refine these features in both domains interactively, makes for a best state-of-the-art method to successfully improve the sensor fusion. Using this strategy, it is possible to maintain and preserve information that might get lost if the interactions between subsequent domains are not modeled and consequently mitigating the large gap between the two domains when performing fusion.

2.4 Outlook and Conclusion

On this chapter we focused on that part of the perception process, the fusion between LiDAR, Radar and Cameras sensors, in order to provide a detailed map of the environment, which is crucial in the field of autonomous navigation. We presented two approaches, CenterFusion and 3D Dual-Query Dual-Fusion. We implemented and reproduced the results of the original respective papers, in order to get a better comprehension of the strategies. We then continued experimenting with the View-of-Delft dataset. We can notably say that the 3D Dual-Query Dual-Fusion makes for a better approach in handling and enhancing the features of both domains before feeding them to the Transformer and fusing the features interactively and simultaneously, in both camera- and voxel-domains.
For later work we recommend to dive into the CenterNet/Track image detection approach and to train a model on the VoD image data and validate the thesis that the pretrained model plays an important role. The resulting models should be used as the pretrained model in der CenterFusion implementation. This is the basis for a well functionating object detection with CenterFusion.
After figuring this out the next step should be to further investigate the quality and precision of the detection. It should be investigated if the detection confidence scores are increased with a correctly trained pretrained model and lead to better results.
Investigating these factors and understanding the CenterFusion implementation are the keys for a high-quality sensor fusion. As CenterFusion is already detecting objects on the VoD dataset, we are very optimistic that CenterFusion has the potential to fully work on the VoD-dataset and can be adapted to any other available dataset.

Disclaimer:

For the purpose of enhancing linguistic clarity, coherence, grammar and overall structure, parts of this chapter have been reviewed and refined using ChatGPT, an artificial intelligence language model developed by OpenAI. While ChatGPT can generate and improve human-like text, it doesn't inherently validate the accuracy or factual content of the report. In 2.1 Overview ChatGPT was used to enhance the linguistic clarity of the section.

3 Environment Modelling with Unreal Engine 5.1

The Automotive industry has witnessed a significant paradigm shift with the emergence of automated driving technologies. The development of sophisticated environment modeling systems that enable vehicles to navigate and make decisions in complex real-world scenarios becomes an important step towards efficiency of this innovation. This work package delves into the realm of environment modeling for automated driving and explores the integration of Unreal Engine 5.1 as a versatile platform for designing, testing, and validating autonomous vehicle systems.
Unreal Engine 5.1 facilitates the creation of varied scenarios, encompassing different weather conditions, lighting scenarios, traffic patterns, and pedestrian behaviors. These scenarios provide a virtual playground for testing and validating the capabilities of automated driving algorithms, contributing to safer and more robust autonomous vehicle technology.
Additionally, implementation of vehicle and pedestrian detection is an important asset in Automated Driving. YOLOv5 can be used for vehicle and pedestrian detection tasks, as it is a versatile object detection model capable of detecting various classes of objects in real-time. The incorporation of the YOLOv5 model for vehicle and pedestrian detection adds an extra layer of authenticity, as precise object detection is paramount for ensuring the safety and effectiveness of autonomous driving systems.
The foundation of this project lies in Dataset which will be used to create realistic adverse weather and lighting scenarios in Unreal Engine 5.1. These scenarios are designed to test the object detection system thoroughly. Achieving this requires the development of 20 unique, static scenarios, each simulating various adverse conditions.

3.1 Unreal Engine and Automated Driving

3.1.1 What is Unreal Engine?

Unreal Engine, developed by Epic Games, is a comprehensive suite of tools and technologies that has evolved from its origins as a game engine. It provides developers with a versatile platform to create interactive and visually impressive applications across a wide range of industries. While initially designed for game development, its capabilities have led to its adoption in areas such as in automotive industries for simulations, architectural visualization, film production, virtual reality experiences, and more.

One of its significant achievements is the high-fidelity rendering capability. Leveraging sophisticated algorithms, the engine simulates the intricate interactions of light with digital entities, rendering scenes that are almost indistinguishable from reality.

With the introduction of Unreal Engine 5 (which is the version used in this project), technological strides like real-time global illumination through "Lumen" and the virtualized geometry system "Nanite" came into the spotlight. The former dynamically adjusts lighting based on environmental changes, while the latter allows incorporation of high-detail art assets without taxing system resources.

3.1.2 Why to use it for creating a simulated environment for Automated Driving

The engine's flexibility extends to the design and customization of scenarios tailored to evaluate various aspects of automated driving. These scenarios encompass tests for obstacle avoidance, lane following, and pedestrian interaction. Such adaptability allows for the crafting of scenarios that target specific use cases and enable the testing of edge cases that may prove challenging to replicate in actual, physical environments.

The engine's built-in physics simulation engine accurately models the interactions between objects, vehicles, and the environment, replicating real-world physics. Simulated environments also offer precise control over various parameters, including lighting, weather, traffic, and pedestrian behavior. This level of control enables consistent test repetition and the isolation of specific conditions for comprehensive testing and validation. The physics engine ensures that the behavior of autonomous vehicles within the simulation aligns closely with real-world physical principles. This is also important for the detection which is carried out in case of pedestrians and other vehicles.

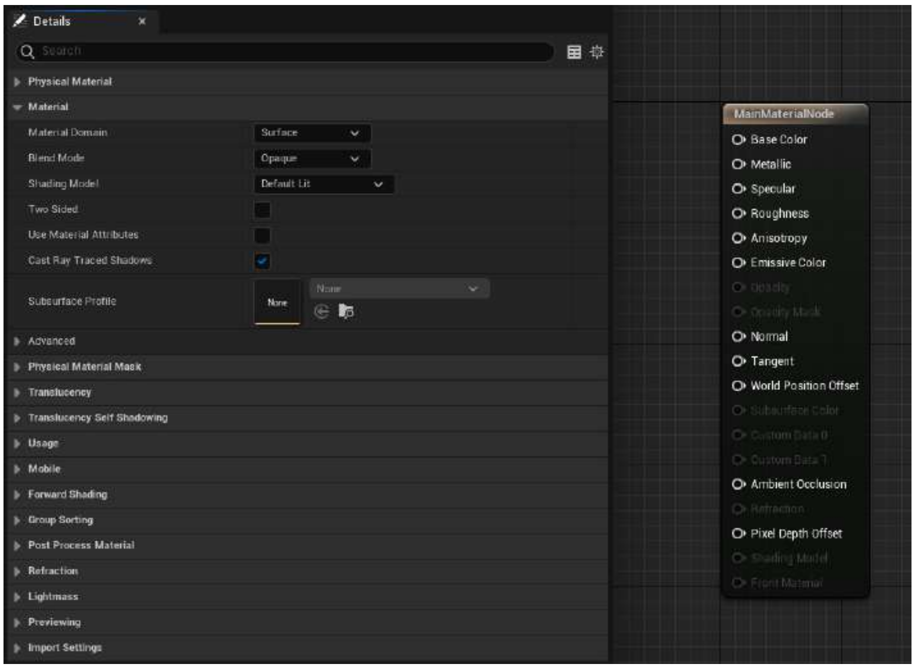
Simulating weather conditions and lighting scenarios is another vital aspect of automated driving testing. Unreal Engine is able to simulate a wide range of weather conditions, from rain and snow to fog and dynamic lighting changes. This capability allows developers to evaluate the performance of sensors, perception algorithms, and control systems under adverse weather and low-light conditions, ensuring the safety and reliability of autonomous vehicles in challenging environments. Unreal Engine's Blueprint visual scripting system and extensive library of assets empower rapid prototyping and iteration of simulated environments. This agility is crucial for keeping pace with the swiftly evolving field of automated driving.

From a cost perspective, utilizing a virtual environment for testing proves to be more economical compared to relying solely on physical vehicles and real-world test sites. This approach reduces expenses related to vehicle wear and tear, fuel consumption, insurance, and the need for physical test tracks. Safety is a paramount concern in the development of autonomous vehicles. Simulated environments provide a secure testing environment where developers can simulate complex and potentially hazardous scenarios without the risk of damaging physical vehicles or compromising human safety.

3.2 Materials in Unreal Engine 5.1

Materials are an integral aspect of creating realistic and visually appealing scenes in Unreal Engine. They serve as the foundation for defining how surfaces and objects within a virtual environment appear and interact with light. Understanding materials is crucial for achieving the desired aesthetics in your game or project.

Unreal Engine provides a powerful tool for working with materials: the Material Editor. This visual interface allows developers and artists to create, modify, and preview materials efficiently. It operates on a node-based system, where you connect various nodes to build the material's structure. These nodes encompass a wide range of operations and properties, from texture sampling to mathematical calculations and material attributes.



Material Domain	Surface
Blend Mode	Opaque
Shading Model	Default Lit
Two Sided	☐
Use Material Attributes	☐
Cast Ray Traced Shadows	✓
Subsurface Profile	None

Figure 3.1: Material Nodes and Functions

Textures play a pivotal role in materials, serving as the basis for various material properties. Texture samples within the Material Editor represent images, such as diffuse colors, normal maps, roughness maps, and more. These textures are essential for determining how light interacts with the surface, influencing aspects like color, reflectivity, and surface roughness.

Mathematical nodes are another vital component of the Material Editor. They enable users to perform operations on inputs, making it possible to blend textures, control material properties, and create complex material behavior. This flexibility empowers creators to craft materials that respond precisely to their artistic vision.

Material properties are central to defining how a material interacts with light. Unreal Engine's Physically-Based Rendering (PBR) workflow ensures that materials mimic real-world behavior. Common properties like Base Color (representing diffuse color), Metallic (controlling reflectivity), Roughness (determining surface smoothness), Normal (specifying surface direction), and Emissive (enabling self-illumination) contribute to this realism.

Material Instances allow for material parameterization and variation. Derived from a base material, instances enable adjustments to specific parameters without necessitating the creation of entirely new materials. This is especially useful when creating variations of materials for different assets or scenarios.

Unreal Engine's material system also handles the generation of shaders. These shaders are essential for GPU rendering and are automatically generated based on the nodes and settings within the Material Editor. They ensure that materials appear as intended when rendered in the game engine.

Material blending and layering provide additional flexibility. By blending multiple materials together, developers can achieve effects such as decals, dirt accumulation, or snow cover. This feature enhances the visual complexity and richness of virtual scenes (*Engine Materials Tutorials | Unreal Engine 5.1 Documentation* n.d.).

3.2.1 PBR Model and textures of Unreal Engine

Physically Based Rendering, often abbreviated as PBR, represents a significant advancement in the field of computer graphics rendering. Unlike traditional rendering techniques that rely heavily on the artistic intuition of setting material parameters, PBR is rooted in the principles of physics and optics. Its primary objective is to simulate how light interacts with surfaces in a manner that closely mimics the behavior of light in the real world. Materials tell the render engine exactly how a surface should interact with the light in your scene and define every aspect of the surface including color, reflectivity, bumpiness, transparency, and so on. These calculations are done using data that is input to the Material from a variety of images (textures) and node-based Material expressions, as well as from various property settings inherent to the Material itself.

One of the fundamental aspects of PBR is the concept of material attributes. These attributes are pivotal in achieving realism and believability in virtual scenes. Among these attributes, the Base Color is a key component. It defines the primary color of a material using the RGB color space, where each channel (Red, Green, and Blue) is constrained to a range between 0 and 1. Essentially, the Base Color sets the base tone of the material, determining whether it appears red, blue, green, or any other color in the visible spectrum.

Another crucial attribute in PBR is Roughness. Roughness controls the degree of rough-

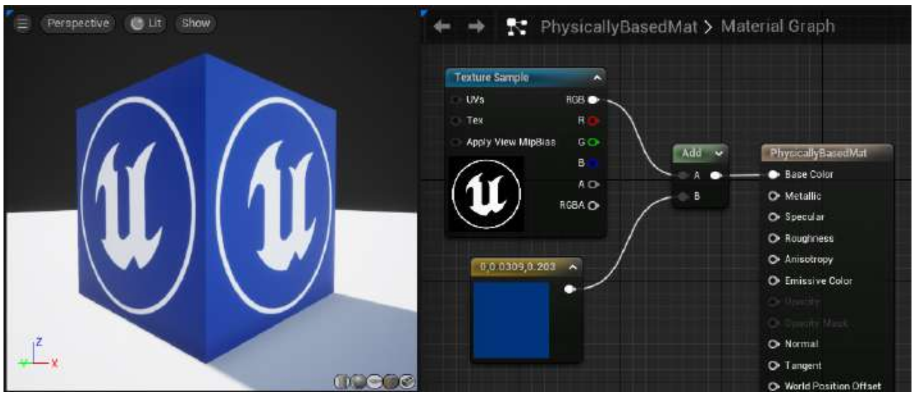


Figure 3.2: Material Graph for the Base Color

ness or smoothness exhibited by a material's surface. A Roughness value of 0 signifies an exceedingly smooth surface, akin to a perfect mirror that produces sharp, mirror-like reflections. On the opposite end, a Roughness value of 1 represents a highly rough or matte surface, where reflections become diffused and blurry.

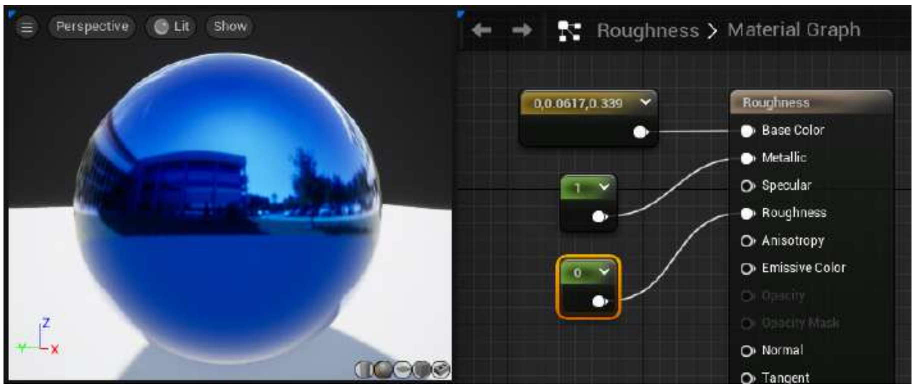


Figure 3.3: Roughness inputs with different values between 0 and 1

The Metallic attribute, in PBR, provides the capability to define whether a material behaves as a metal or nonmetal. This attribute accepts values between 0 and 1. A value of 0 designates a non-metallic material, such as plastic or wood, while a value of 1 characterizes a fully metallic material like copper or steel. Metallic materials typically exhibit strong specular reflections, which manifest as bright, mirror-like highlights on their surfaces. Non-metallic materials tend to have lower metallic values, leading to subdued or absent specular highlights.

Lastly, the Specular attribute controls the intensity of specular, or mirror-like, reflections on the material's surface. Specular reflections are the bright spots or highlights observed on shiny surfaces when they interact with light sources. A Specular value of 0 implies that the material is entirely non-reflective, lacking these highlights. In contrast, a Specular value of 1 denotes that the material is highly reflective, resulting in pronounced specular highlights.

In combination with other material properties such as Normal Maps (for simulating surface

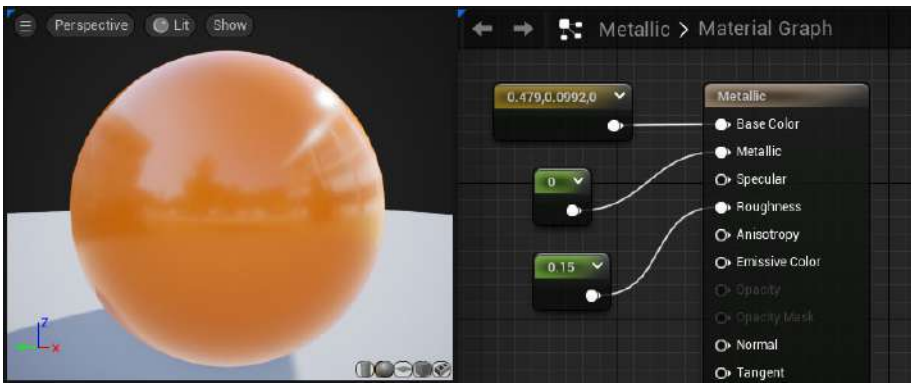


Figure 3.4: Metallic inputs with different values between 0 and 1

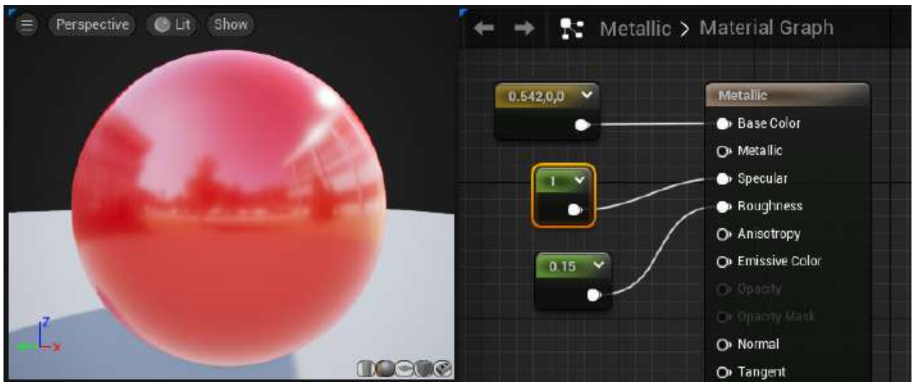


Figure 3.5: Specular inputs with different values

details), Ambient Occlusion (for realistic shading in crevices), and Emissive Color (for materials that emit light), PBR has become a cornerstone in designing simulations for Automated Driving. By faithfully modeling the physics of light interaction, PBR enables the creation of captivating and realistic 3D environments, elevating the quality and immersion of digital experiences to unprecedented levels (*Engine 5.1 Documentation* n.d.).

3.2.2 Creation of Adverse weather materials

Creating Snowfall effect: To begin replicating snowy conditions in Unreal Engine 5.1, a dedicated Snow Material needs to be created. This material is responsible for giving surfaces that distinctive snow-covered appearance. Fine-tuning various material properties to mimic snow buildup is done. Utilizing texture maps, such as normal maps and height maps, can be effective for simulating the texture and depth of snow on objects. Adjusting opacity settings which determines how much of the underlying material is obscured by the snow is crucial as well. To complete the snowfall effect, consider implementing particle systems using Unreal Engine's Niagara or Cascade systems, where parameters like snowflake size, speed, and density are controlled like in Figure 3.6.

Creating Rainfall Effect: Creating a Rain Material follows a similar process as crafting the snow counterpart. A process is initiated by creating a new Material in Unreal Engine 5.1. The primary objective here is to make surfaces look wet and reflective, closely re-

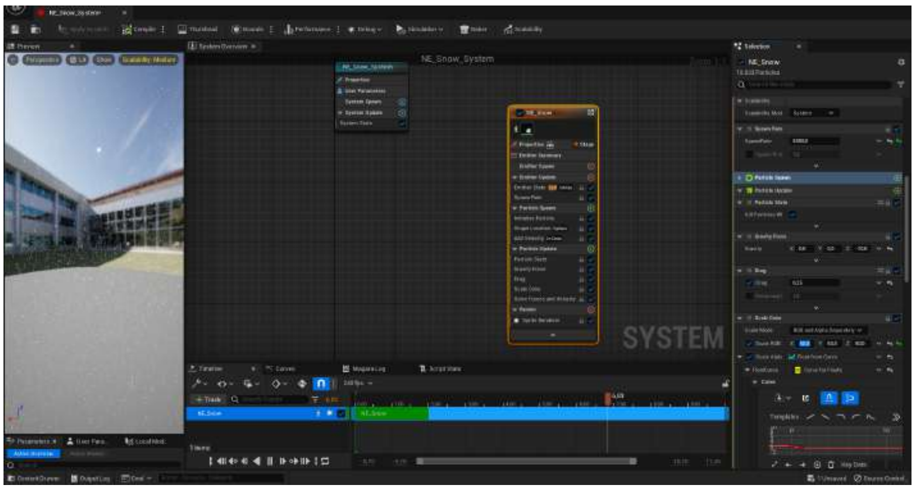


Figure 3.6: Material Graph of the Niagara System for Snowfall effect

sembling rain-soaked conditions. Normal maps can be particularly useful for mimicking water droplets running down surfaces. By modifying material properties, you can achieve the desired visual effect. Additionally, you can employ particle systems to simulate falling raindrops. Customize particle parameters such as size, speed, and density to replicate realistic rainfall within the scene as shown in Figure 3.7.

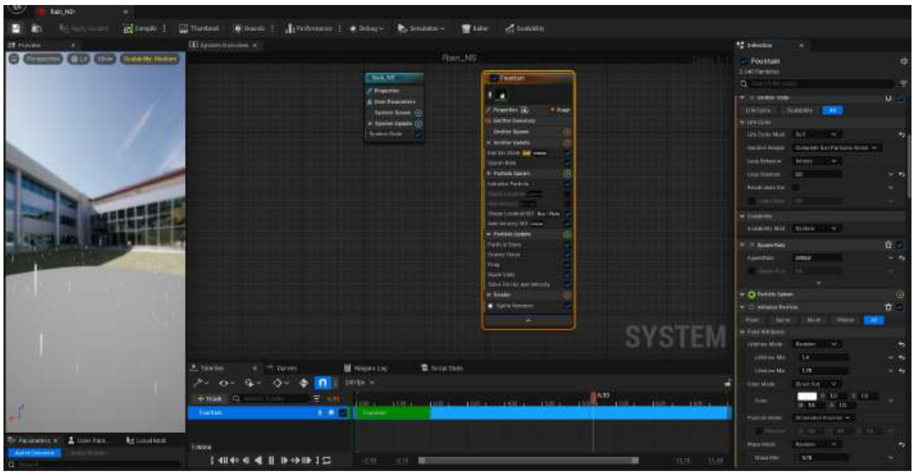


Figure 3.7: Material Graph of the Niagara System for Rainfall effect

Implementing Fog: For introducing fog and atmospheric effects into the Unreal Engine 5.1 project, several options are available. The Atmospheric Fog actor is used for this. By configuring properties like Fog Density, Color, and Height Falloff, you can create diverse fog effects, ranging from light mist to dense, impenetrable fog. The Exponential Height Fog actor is another option to consider, allowing to manipulate settings like Fog Density, Fog Inscattering Color, and Fog Height Falloff to influence the fog's appearance and behavior. Parameters like Density, Lighting, and Scattering can be adjusted to craft realistic volumetric fog. Fog effects can be refined further by complementing these actors, by employing post-process volumes. Adjust settings such as Bloom, Depth of Field, and Color Grading to fine-tune the fog's appearance in the scene. Additionally, fog material can be created to have different densities of fog for different scenarios. Installing a plane and applying this material to it, will provide with requirement specific foggy natures as shown in Figure 3.8.

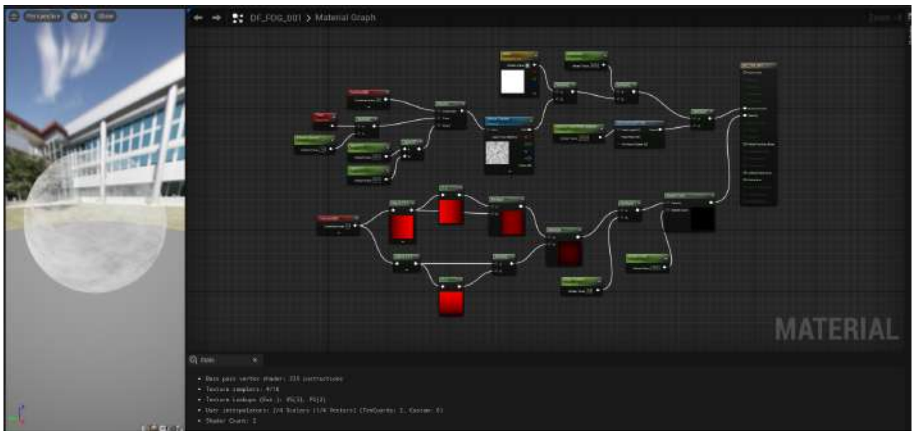


Figure 3.8: Material Graph of the Fog material

3.3 Recreation of scenes from The Adverse Weather Dataset

Recreating scenes from The Adverse Weather Dataset within Unreal Engine 5.1 is a process that requires a methodical and detail-oriented approach. To achieve this, it is essential to break down the process into various key components, starting with a comprehensive analysis of the dataset scenes. This analysis involves a thorough examination of the terrain, weather conditions, lighting, assets, and overall scene layout. Understanding the specific characteristics and components of the scenes is fundamental for an accurate recreation.7

The first step is to lay out the main roads and the sidewalks using splines, and applying the asphalt surface texture for roads or concrete material for sidewalks and adding the road markings if needed. After that we place the main object or vehicle which is the base for the camera (scene point of view). The next step involves asset preparation. Proper asset placement is essential for ensuring that the scenes are complete and faithful to the source material. (see Figure 9) This includes collecting the necessary 3D assets, such as buildings, vehicles, vegetation, and other objects that are featured in the dataset scenes. High-quality 3D assets are fundamental for achieving a high level of realism in the recreated scenes. Sometimes if the 3D assets cannot be acquired in a way that is similar to the original scene, a cube or sphere or a combination to make the shape similar to the 3D asset can be added in the original scene.

Terrain and environment play a significant role in scene recreation and detection. There are robust landscape tools that enable the sculpting of terrain to match the geographical features observed in the dataset scenes. Additionally, various surface textures representing different types, such as grass, asphalt, or rock, concrete are carefully applied to the terrain. These textures not only enhance visual realism but also add depth and authenticity to the recreated environment.

The adverse weather conditions that are a defining feature of the dataset scenes are simulated using Unreal Engine's powerful particle system tools. Rain, snow, fog, and other atmospheric elements are dynamically brought to life through these particle systems. One of such systems is the Niagara particle System. Furthermore, asset materials undergo adjustments to accurately reflect the visual impact of weather conditions. This includes creating the appearance of wet surfaces during rainfall, foggy atmosphere, scattering of the sunlight to blur the scene, etc. contributing to the overall realism of the scenes.

Many of the dataset scenes include intricate road networks and pathways. Unreal Engine's spline tools prove to be invaluable for creating these roads and paths, allowing for the incorporation of curved and complex layouts. In the scenes present in the dataset, the paths and roads are mostly straight. So, in order to avoid complexity, flattened cubes can be used as roads instead of splines. To ensure authenticity, road-specific textures and markings are meticulously applied, aligning them with the scenes' characteristics and enhancing their overall realism.

Lighting optimization is crucial for achieving visual realism. Unreal Engine 5.1 offers a range of light actors, each serving distinct purposes. For instance, Directional Light simulates sunlight and can be adjusted in terms of intensity, color, and direction. Furthermore, the directional light location in the sky emulating the sun can represent different times of the day like sunset, sunrise, dawn, night and day. Point Lights emit light in all directions from a single point and are commonly used for indoor scenes and various lighting effects. Spotlights, on the other hand, emit light in a cone-shaped direction and are valuable for creating focused and directed light sources, such as flashlights or stage lighting. Road Lamps are added which already have light emissive material, but sometimes spotlights are added to the lamp location to increase the intensity. The sky and atmosphere are vital elements in scene recreation, and Unreal Engine simplifies their simulation. The engine provides built-in components like Sky Atmosphere and Sky Light, enabling the creation of atmospheric effects. Post process volume is finally added to the scene to process the exposure, bloom, and light flare.

Material shaders play a pivotal role in replicating lighting and weather effects within the recreated scenes. These shaders can be employed to simulate a range of conditions, including wet surfaces after rain, snow accumulation, or even the flickering of flames. Material shaders add depth and realism to the scenes, enhancing their visual fidelity.

To optimize performance, particularly for scenes with complex geometry, Level of Detail (LOD) techniques are applied to more intricate 3D models. These techniques help minimize resource usage while preserving visual quality, ensuring that the simulation runs smoothly and efficiently. Throughout the entire process, scenes should undergo continual review and iteration. This involves meticulous examination for accuracy and realism, with adjustments made as necessary based on feedback and observations. The iterative process is instrumental in refining the scenes and ensuring that they closely match the source material from The Adverse Weather Dataset

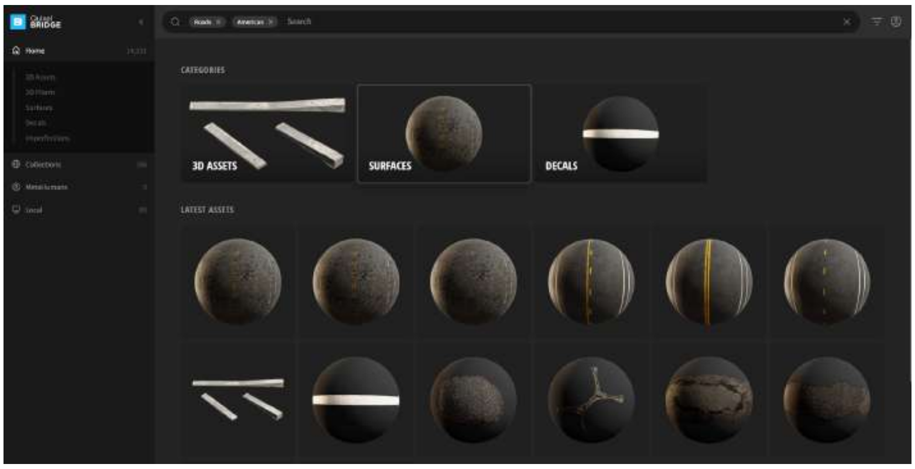


Figure 3.9: Example of 3D Assets available in Quixel Bridge

3.3.1 Lighting conditions/parameters in Day and Night scenes

Parameters that need to be adjusted for the day scenes:

In the realm of 3D scene rendering, the Directional Light serves as a pivotal element for achieving realistic lighting effects. Its intensity is a fundamental factor; for daylight scenes, cranking up the intensity is essential to cast well-defined shadows and simulate the brilliance of the sun or sunset. Conversely, in overcast settings, a lower intensity setting is preferable to create a soft, diffused lighting ambiance. The color temperature of this light source can be meticulously tuned to match the precise requirements of the scene, offering control over the warm or cool hues of the light, which can profoundly impact the overall mood. Additionally, the angle at which the directional light is positioned can be adjusted, allowing for the faithful reproduction of natural lighting conditions, including the creation of shadows and highlights that mimic real-world scenarios.

On the other hand, the Sky Atmosphere settings play a crucial role in crafting the backdrop of the scene. The Atmosphere Height parameter comes into play here, enabling the alignment of the virtual cloud layer with the desired environmental conditions, enhancing the scene's authenticity. Multi-scattering, another parameter, governs the rate at which sunlight interacts within the atmosphere, influencing the scattering behavior of light, which, in turn, affects the level of diffusion and atmospheric interaction. Rayleigh Scattering takes its cue from the atmosphere's density, shaping the sky's color palette; higher values introduce a broader range of colors, fundamentally altering the atmospheric hue. Meanwhile, the Rayleigh Exponential Distribution parameter lets you control the position of the Rayleigh scattering falloff, where lower values translate to a darker sky, while higher values usher in novel color variations in the sky, diversifying its visual appeal.

In parallel, the Mie Scattering parameters focus on the interaction of light with spherical or cylindrical objects, like our planet Earth. The Mie Scattering Scale impacts the atmosphere's density due to Mie scattering, making it appear thinner and closer to the Earth's surface, especially around the sun and near the horizon. Adjustments to the Mie Scattering parameter itself influence the atmospheric coloration, contributing to the overall sky appearance. Furthermore, the Mie Absorption Scale governs the extent to which light is absorbed due to Mie scattering, affecting light attenuation as it traverses the atmosphere. Finally, Mie Absorption is linked to the color absorbed by the light, enabling alterations in the sky's color; for instance, absorbing blue light can result in a reddish sky and vice versa. These parameters collectively empower creators to finely tune lighting and atmospheric conditions, offering the means to construct diverse and lifelike virtual environments that resonate with real-world settings.

Parameters that need to be adjusted for the night scenes:

Crafting a convincing night scene in Unreal Engine involves a meticulous and multi-faceted approach that encompasses various critical considerations, ranging from the intricacies of lighting to the nuances of post-processing. These key elements come together to create an authentic and immersive nocturnal setting.

Firstly, the Directional Light plays a pivotal role. To establish the desired ambiance of a night scene, it's imperative to reduce the intensity of the light source significantly. This dimming effect is central to capturing the subdued atmosphere associated with nighttime. Furthermore, adjusting the color temperature of the light source is essential. Typically, this involves shifting the light towards cooler, bluish tones or desaturating it entirely to simulate moonlight or the absence of the sun's warm hues. The angle of the light is another crucial consideration, as it determines the direction from which moonlight or other nocturnal illumination sources emanate, ensuring the scene's authenticity.

In parallel, the Sky Light component requires thoughtful adjustment. Its intensity should be considerably lowered to align with the reduced levels of ambient light in a night scene. The color of the Sky Light can either be kept neutral or subtly shifted towards bluish tones, reinforcing the cool, tranquil tones characteristic of the night.

Post-process effects play a vital role in enhancing the overall look and feel of a night scene. Exposure control is a fundamental aspect, regulating the camera's sensitivity to available light. In the context of a night scene, it's essential to reduce exposure significantly, imparting a sense of darkness and intimacy. Managing bloom, a visual effect that mimics the gentle glow around bright objects, is crucial. While lowering bloom intensity is generally required to replicate the subdued reflectivity of nocturnal surfaces, some controlled bloom may be desirable for specific light sources, such as streetlights or neon signs. Lastly, lens flares can be artistically integrated to simulate the optical phenomena caused by bright nocturnal light sources like streetlights or vehicle headlights, adding an extra layer of authenticity and atmosphere to the scene.

3.3.2 General Method for Scene Recreation

Recreating scenes in Unreal Engine 5.1 for automated driving simulations involves several steps, including creating 3D environments, adding static or dynamic meshes, importing assets, setting up materials, adjusting the parameters such as weather and lighting conditions, implementing traffic and pedestrian positions, camera position and then optimization.

The initial step to creating a scene is to form a Basic Level inside a Main Project. All the scenes are made on different levels but in the same project. Creating different projects for different scenes is tedious as it requires importing of 3D Assets and materials from Quixel Bridge or Unreal Engine Marketplace in every project, thus increasing the storage. For creating real-world scenes, it is better to import the materials and textures which are required beforehand.

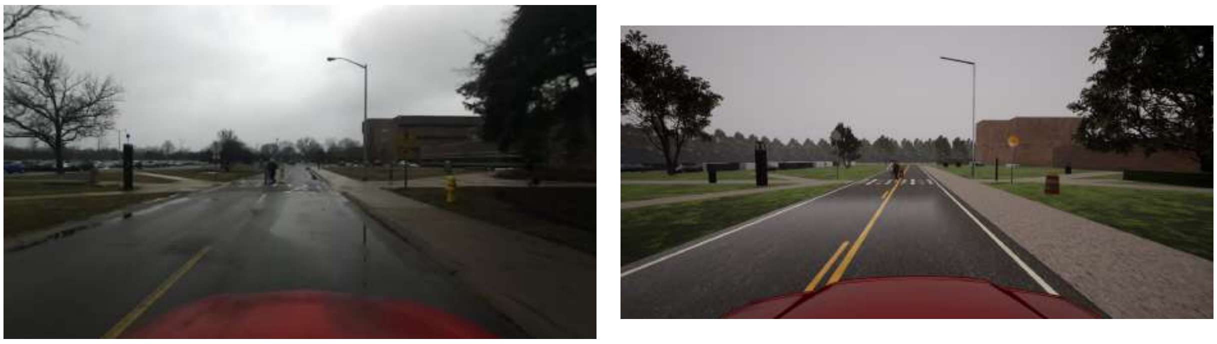


Figure 3.10: Construction of the scene from Dataset and description of its recreation

The recreation starts with building a landscape in the Landscape Mode in Windows Toolbar. The Basic Level has a limited working space and attaching a Landscape over it will extend the workspace. Comparing the scenes, deciding on the material of the landscape is important as it will give an overall complexion to the scene. Here, Green Grass material is chosen from the Content Browser and applied to the landscape. Refinement of the material entails meticulous adjustment of its properties to harmonize with the original scene. This adjustment predominantly involves fine-tuning the color attributes of the base layer of the material to achieve the desired visual aesthetics. The progression then goes into the meticulous construction of roads and footpaths within the landscape, a pivotal aspect of scene development. This construction can be executed through two principal methodologies:

- **Spline-Based Construction** When the design specifications necessitate curved roads or pathways, the judicious employment of splines is advisable. Splines facilitate the creation of smooth, sinuous trajectories, ideal for replicating intricate or serpentine routes.
- **Cube-Based Construction** Conversely, for the establishment of straight roads or footpaths, cubes emerge as a pragmatic choice. These cubic entities are meticulously tailored in terms of their dimensional attributes, namely, the X-Axis and Y-Axis coordinates, conformed to the requirements of the original dataset. The height or

the Z-Axis coordinates of the cube is minimized, often to a minute value such as 0.01, rendering it a flat road surface that seamlessly integrates with the landscape.

The key determinant for selecting between these methodologies hinges upon the particular demands of the project. Splines offer a sophisticated solution for crafting intricate, curved pathways, whereas cubes are better for their efficiency and simplicity when dealing with linear or straightforward road configurations. In this specific context, we employ Cube-Based Construction as the method of choice due to the absence of a requirement for curved pathways within the level. For the construction of roads, a seamless integration process is initiated, where Fine American Road materials are sourced from Quixel Bridge and applied as the foundational material for the cubic structures. Likewise, for the creation of footpaths, we opt for the application of Concrete Pavement materials to achieve the precise appearance desired. Additionally, the buildings in the scene are prepared from adjusting the X,Y,Z - Scale Coordinates of a cube and adding a similar coloured material as the original scene.

The subsequent phase in the recreation entails the strategic placement of additional materials and 3D assets. This crucial step is significantly streamlined by the collaboration between Quixel Bridge and the Unreal Engine Marketplace, both of which serve as invaluable repositories for acquiring the requisite assets. Within the scope of the current scene, key assets such as the 3D People Pack and City Sample Vehicles Pack have been sourced from the Unreal Engine Marketplace. These comprehensive packs provide an extensive array of options, encompassing a diverse range of vehicles essential for scene construction and object detection, as well as a rich variety of pedestrian elements. Notably, these packs offer the added advantage of allowing for dynamic color adjustments to the assets, enabling the precision needed for scene recreation and efficient object detection processes. Moreover, incorporating trees into the scenes presents two distinct methodologies, each carrying its own set of merits and considerations. The first approach revolves around the use of Popcorn Trees. While they offer a resource-efficient solution, they may not deliver the level of intricate detail and realism sought after for a scene. The alternative method entails the acquisition and integration of tree materials or geometries procured from the Unreal Engine Marketplace. In all the scenes of this project, tree geometries curated from resources such as the Black Adler Pack and the European Hornbeam Trees Pack are used. These crafted assets have played an instrumental role in the backdrop of the specific scene. The disadvantage is that the utilization of imported assets does carry the trade-off of increased storage consumption.

The subsequent phase entails the fine-tuning of lighting conditions within the scene. Each scene predominantly comprises essential elements such as the Directional Light, Sky Sphere, Exponential Height Fog, Sky Atmosphere, Sky Light, and Volumetric Cloud, all of which are adjusted in accordance with the specific scene requirements. In the depicted scene 3.10, the prevailing weather conditions exhibit a cloudy ambiance, though not as dark as a night scene. Within this context, the Directional Light plays a pivotal role in the overall illumination of the scene. To achieve the desired cloudy sky effect, alterations are made to the density and intensity parameters of the Volumetric Cloud. Furthermore, properties such as cloud height and distance are modulated via these adjustments. Additionally, critical parameters influencing the scattering of Directional Light across the scene, namely Mie Scattering and Rayleigh Scattering, are manipulated via the Sky Atmosphere settings. Specifically, for the given scene, their values fall within the range of 1.0 to 2.0, facilitating comprehensive light scattering and contributing to the scene's foggy ambiance.

The Exponential Height Fog parameter governs the intensity and density of the fog. Given the scene's relatively low fog density, this parameter is maintained within the range of 0.0 to 1.0. It is worth noting that scenes with a heavier fog requirement, such as snowy environments, may necessitate higher fog density settings, which can be adjusted to align with specific scene demands.

The Sky Sphere and Sky Light parameters focus on the hue of the sky. The Sky Light Actor is a component used to capture the distant parts of the level's sky and use it to provide ambient lighting for the scene. It simulates the scattering of light by the atmosphere and can contribute to the overall lighting in your level. The Sky Sphere is placed in the level and is usually positioned around the viewpoint to provide a visually appealing sky and to create dynamic changes in lighting and sky visuals.

The next stage in the process involves the incorporation of Puddle Layers onto the road surface to emulate the wet road characteristics present in the original dataset scene. This task is executed with the aid of Mesh Painting mode. To initiate, a specialized Blend Material must be meticulously crafted within Quixel Bridge by fusing together two or three distinct layers. It's customary to select the uppermost layer, which, in this case, corresponds to the American Road Material layer, as the primary layer for painting. The remaining two layers do not carry any specific precedence. Once the blended material is formulated, it becomes imperative to activate all checkboxes located within the global details tab. This ensures the development of a high-quality blended material. A critical note here is that the option associated with the Puddle Layer must be enabled before applying the material. Failure to do so will result in the absence of puddles on the designated layer. A transition from the Selection Mode to Mesh Painting mode is executed via the Toolbar. The 'Paint' functionality is then chosen to enable the application of paint to the predetermined areas. Following this, under the Vertex Painting tab, the Paint Color and Erase Color settings should be configured to Black, and the Color Painting tab should have the Blue Channel selected. This configuration effectively activates the puddle-painting mode, granting the ability to precisely apply and manipulate puddles on the road surface. Notably, parameters such as radius, depth, and intensity of the puddles can be adjusted to achieve the desired effect.

The ultimate phase in scene recreation encompasses the integration of supplementary assets that may potentially impact object detection. For instance, within numerous scenes, sidewalks harbor assorted elements such as traffic poles and signs, small bushes and trees, and more. Additionally, within this specific scene, a parking lot with numerous vehicles is visible. However, not all vehicles are consistently detected due to inherent limitations within the model. Furthermore, the inclusion of vehicles from the City Sample Vehicles

Pack, while feasible, raises concerns about substantial storage consumption, potentially inducing performance degradation or system crashes. A viable alternative entails the utilization of cuboids that simulate automobiles, strategically positioned within the scene. Notably, the default hue of these cuboids is white, but their color can be conveniently customized to align with specific requirements.

The culmination of the above steps involved in scene recreation, encompassing the adjustments to directional lighting, atmospheric conditions, and various other parameters, demonstrates the capacity for these techniques to be universally applied with precision. By adhering to these protocols, one can successfully achieve an imitation of the original scenes featured in the Adverse Weather Dataset. Furthermore, when subjected to detection processes within these recreated environments, Unreal Engine 5.1 emerges as an exceptionally robust platform, adept at simulating the intricate landscapes required for Automated Driving scenarios. This robustness underscores the potential of Unreal Engine as a formidable tool for the development, testing, and validation of autonomous driving systems. In essence, this approach offers a pragmatic, secure, and cost-effective avenue for exploring practical applications in the real-world context of autonomous driving.

3.3.3 YOLOv5 Architecture

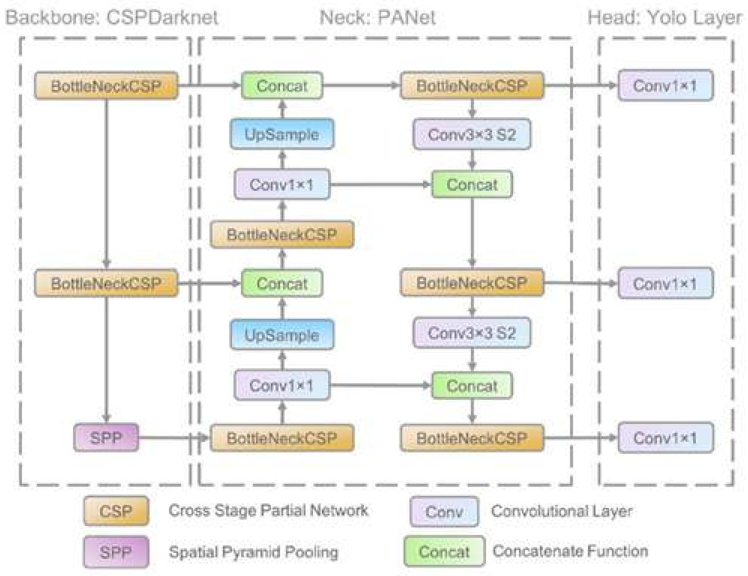


Figure 3.11: Architecture of a YOLO network

The YOLO (You Only Look Once) network is a revolutionary object detection algorithm that has gained popularity for its ability to perform real-time object detection with impressive accuracy. Its success can be attributed to its unique architectural design, which consists of three main components working in concert to detect objects within images or video frames.

Firstly, there's the Backbone of the YOLO network. This is typically a convolutional neural network (CNN), and it plays a fundamental role in the initial stages of object detection. The backbone's primary task is to process the input image and extract features from it at different scales. It essentially breaks down the image into various components, identifying edges, textures, shapes, and other visual patterns.

Following the backbone, there is the Neck, which serves as an intermediate component within the YOLO architecture. The purpose of the neck is to take the features generated by the backbone and further enhance them. This is done through a series of additional layers that combine and mix the features. The neck captures contextual information, allowing the YOLO network to gain a deeper understanding of the relationships between different parts of the image.

Finally, there is the Head of the YOLO network. The actual detection of objects happens here. The head consumes the processed features from the neck and performs the last critical steps of the object detection process. It accomplishes two main tasks: predicting bounding boxes and assigning class labels. These bounding boxes indicate where the objects are located. Additionally, the head assigns class labels to the detected objects to specify what they are, such as "car," "dog," "person," and so on. These predictions are made based on the processed features and form the final output of the YOLO network (Ding, Li, and Yastremsky 2021).

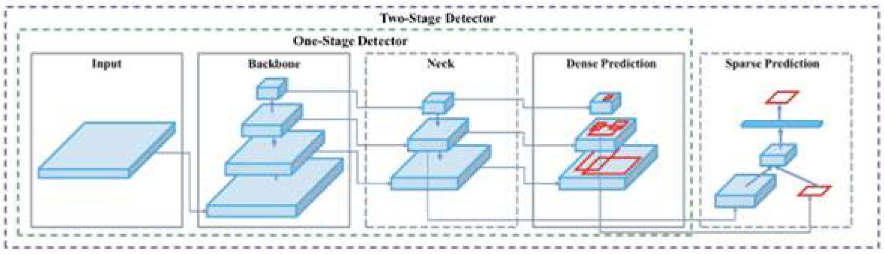


Figure 3.12: Single Stage Detector Architecture

Object detection, a use case for which YOLOv5 is designed, involves creating features from input images. These features are then fed through a prediction system to draw boxes around objects and predict their classes. The process of object detection using YOLOv5 starts with an input image that encapsulates the scene where vehicles and pedestrians are to be identified. This image can vary in dimensions, accommodating different scenarios.

For feature extraction, YOLOv5 relies on a convolutional neural network (CNN) as its core architecture. The primary role of this CNN is to discern and extract meaningful features from the input image. To cater to the detection of objects of varying sizes within the image, YOLOv5 often employs a feature pyramid network (FPN).

The core of YOLOv5's object detection capability lies in the prediction of bounding boxes. The model divides the extracted feature maps into a grid consisting of cells. For each cell within this grid, the model predicts multiple bounding boxes, each representing a potential object location. These bounding boxes are associated with several key attributes. Firstly, there is the Objectness Score. This score reflects the model's confidence in the presence of an object within the predicted bounding box. A higher objectness score indicates a higher level of confidence in the existence of an object at that specific location. Next, YOLOv5 predicts "Class Probabilities" for various object categories, such as cars or pedestrians. These probabilities play a pivotal role in accurately classifying the detected objects into specific categories, ensuring precise object recognition. Lastly, the model predicts the "Bounding Box Coordinates" relative to the cell in which it is located. These coordinates encompass the bounding box's position (x, y) and its dimensions (width, height). These predictions are essential for precisely localizing the detected objects within the image.

Following the prediction of multiple bounding boxes and their associated scores, YOLOv5 applies a technique known as non-maximum suppression (NMS). NMS is a critical step in refining the detection output. It eliminates duplicate or low-confidence detections, ensuring that only the most reliable and non-overlapping bounding boxes are retained in the final results. The ultimate outcome of YOLOv5's detection process is a comprehensive list of bounding boxes. Each bounding box is paired with class labels and confidence scores. Typically, these bounding boxes are superimposed onto the input image, creating a visual representation of the detected objects and their respective positions within the scene. Users also have the flexibility to set a confidence threshold, allowing them to determine the minimum level of confidence required for a detection to be considered valid. This threshold can be adjusted to balance precision and recall, tailoring the detection results according to specific needs and confidence criteria (Bochkovskiy, C. Wang, and Liao 2020).

Intersection over Union (IoU): Intersection over Union (IoU) is a fundamental metric used to evaluate the accuracy of object bounding box predictions in computer vision tasks, such as object detection. It measures how well a predicted bounding box aligns with the actual object's location. IoU quantifies the overlap between the predicted bounding box and the ground truth bounding box of the object.

The IoU score is calculated by taking the area of intersection between the predicted and ground truth bounding boxes and dividing it by the area of their union.

A higher IoU score indicates a better alignment between the predicted and ground truth bounding boxes, suggesting that the model's detection is more accurate. Typically, a threshold IoU value is set, and predictions with IoU scores below this threshold are considered false positives or inaccurate detections. IoU is a key component in various post-processing steps, such as non-maximum suppression (NMS), where redundant or overlapping bounding boxes are removed to retain only the most confident and accurate detections.

Confidence Score: The confidence score, often referred to as the "objectness score" in models like YOLO (You Only Look Once), is an essential component of object detection. Each predicted bounding box generated by the model is associated with a confidence

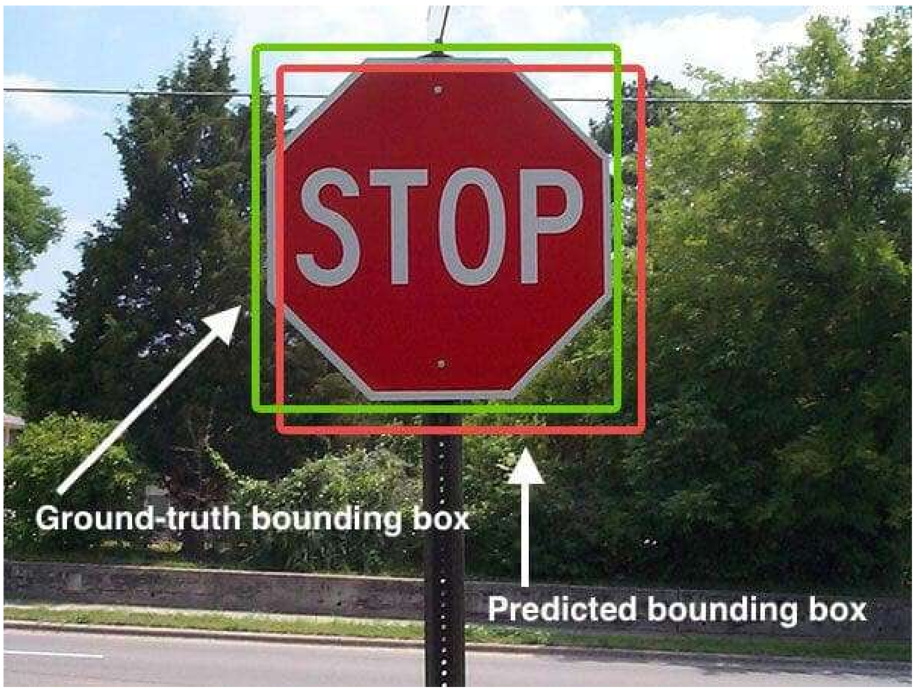


Figure 3.13: Construction of bounding boxes Intersection Over Union (*Intersection over Union (IoU) for object detection* 2023)

score. This score represents the model's confidence in the presence of an object within that specific bounding box.

The confidence score is typically a probability value that ranges from 0 to 1. A high confidence score indicates that the model is very confident that an object is present within the bounding box, while a low score suggests lower confidence in the detection.

During the post-processing stage, when multiple bounding boxes are generated for potential objects in an image, these confidence scores are used to filter and prioritize the detections. In many object detection algorithms, a threshold is set for the confidence score, and boxes with scores below this threshold are discarded as they are considered less reliable.

3.3.4 Impact on YOLO model while changing parameters

Controlling lighting conditions is a pivotal aspect of recreating scenes within Unreal Engine, particularly when it comes to object detection accuracy using models like YOLOv5. Unreal Engine offers good control over lighting, allowing adjustments to factors such as intensity, direction, and color. These alterations can significantly influence how objects within the scene appear, thereby impacting the model's ability to accurately detect them. Consistency in lighting conditions is paramount, as deviations can lead to decreased detection accuracy. It is crucial to ensure that the recreated scenes closely match the lighting conditions under which the YOLOv5 model was initially trained. Maintaining such consistency is vital for reliable object detection outcomes.

The complexity of the environment within Unreal Engine scenes is another crucial factor that can affect object detection. Environments can vary widely in complexity, featuring varying levels of detail, clutter, and occlusions. More complex scenes may present challenges for the model in identifying objects amidst clutter or when objects are occluded by other elements. Achieving appropriate object detection necessitates aligning the scene's complexity with the training data used for the YOLOv5 model. This may involve fine-tuning the recreated scene to avoid overly complex material overlaps and ensuring that it remains within the model's recognition capabilities.

Resolution and graphical quality settings in Unreal Engine have a direct impact on object detection. These settings determine the clarity and level of detail in the images or frames observed by the YOLOv5 model. Lower resolutions or reduced graphical quality can result in a loss of crucial visual information, leading to decreased detection accuracy. To ensure appropriate object detection, it is essential to maintain a resolution and quality level that provides the necessary visual information for the model to perform precise object detection.

Unreal Engine's physics and collision simulation can introduce complexities for object detection. Objects within the scene may occlude one another or undergo unpredictable changes in position due to these interactions. To ensure appropriate object detection, it is advisable to optimize the physics and collision settings within the engine. The goal is to minimize interference with object detection and to ensure that the model can effectively handle occluded or partially visible objects.

Camera parameters, including field of view (FOV) and focal length, are critical variables that influence how objects appear in rendered images or frames. These parameters can significantly impact the model's ability to estimate object sizes and positions accurately. To achieve appropriate object detection, it is crucial to align the virtual camera's parameters within Unreal Engine with those used in the original Adverse Weather Dataset scenes. Adjustments may be necessary to ensure that the virtual camera's position and settings match the model's expectations, facilitating precise object detection.

3.3.5 Examples of the created scenes in Daylight:

Scene 1

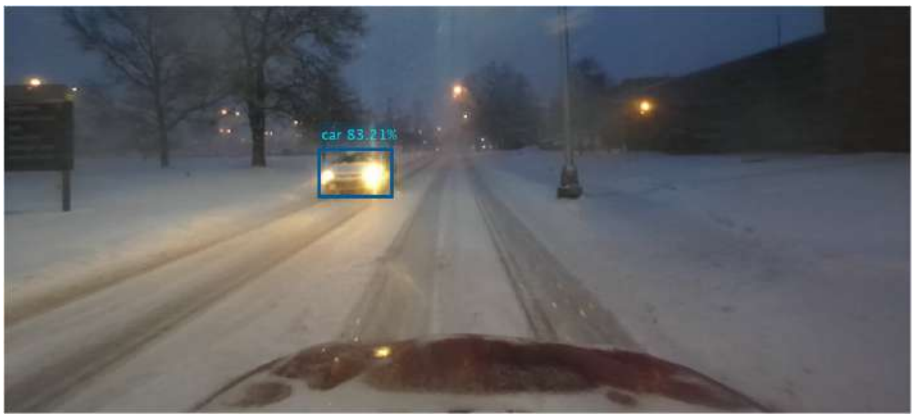


Figure 3.14: Original Image Detection from the Adverse Weather Dataset

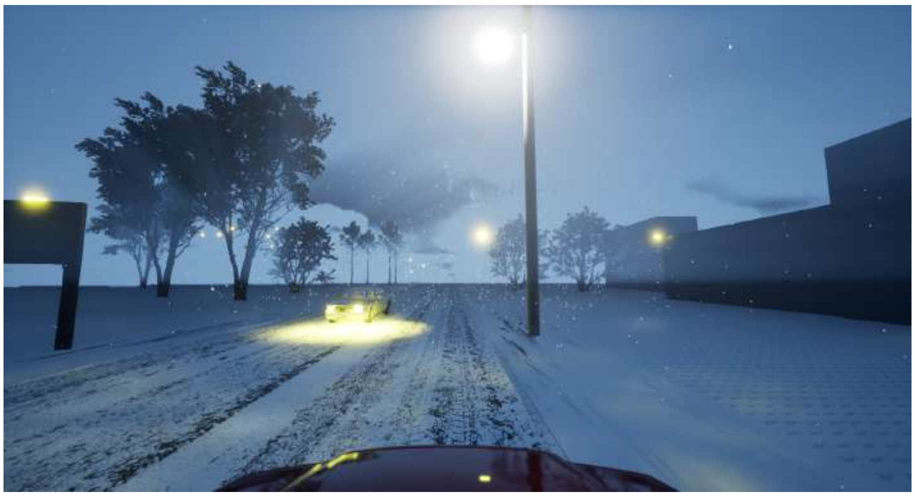


Figure 3.15: Recreated Scene in Unreal Engine 5

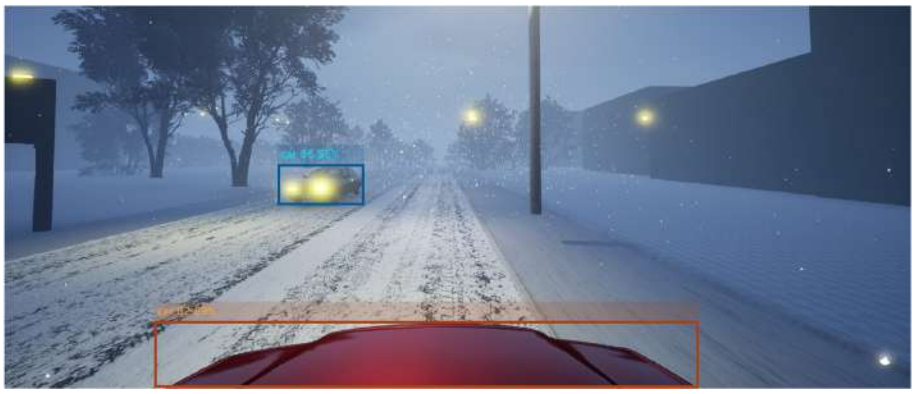


Figure 3.16: After editing the parameters in the scene

Observations and updations

1. **Height of the Cloud Layer and Its Density:** In Figure 3.15 and Figure 3.16, variations in the height and density of the cloud layer are apparent. When clouds are thicker or closer to the ground (higher density), they can diffuse and scatter sunlight. For YOLO, which relies on identifying patterns and edges in images, changes in lighting due to cloud cover can pose challenges. The height is changed here to a lower level and the density is increased so that the scattering of the Directional Light is more near ground level.

2. **Amount of Fog:** The presence of fog is another atmospheric condition that can vary between Figure 3.15 and Figure 3.16. The fog intensity is more in the updated image as before the detection was getting easier because of less density. After increasing the density, the recreated scene matches the foggy nature of the scene from the dataset.

3. **Cone Angle and Intensity of Headlights:** The differences in cone angle and intensity of headlights are likely related to the artificial lighting within the scenes. These parameters influence how far and how brightly the headlights illuminate the surroundings. A wider cone angle can result in more even lighting, while a narrower angle may create concentrated beams of light. Intensity affects how well objects are lit. In Figure 3.15, the cone angle of the street light is not defined which was changed in Figure by incorporating a Spot Light. The new scene describes a well lit spread of the street light. Changing the intensity (see Figure 3.16) shows variation in the YOLO model and gives us detection close to the original dataset.

4. **Collective Impact of Parameters:** Various parameters, including Rayleigh scattering, directional light, sky atmosphere, sky light, Mie scattering, distance between objects, color gradients, shadows, and adverse weather materials, collectively influence object detection with YOLO. For example, shadows cast by objects can change in appearance due to the angle and intensity of light.

Scene 2

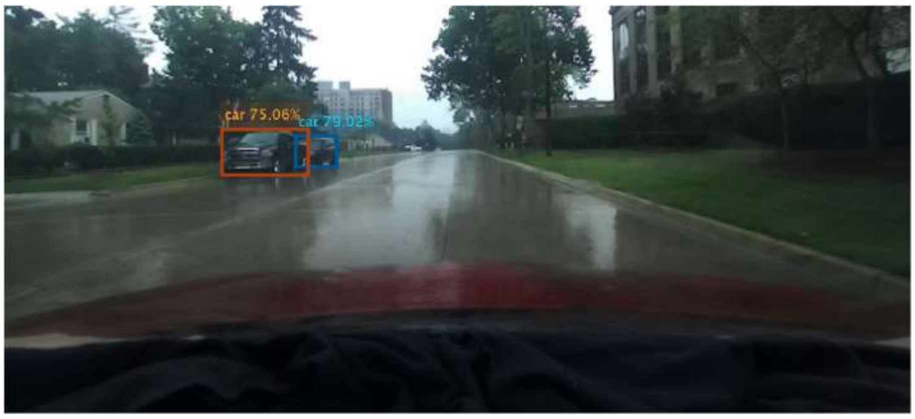


Figure 3.17: Original Image Detection from the Adverse Weather Dataset

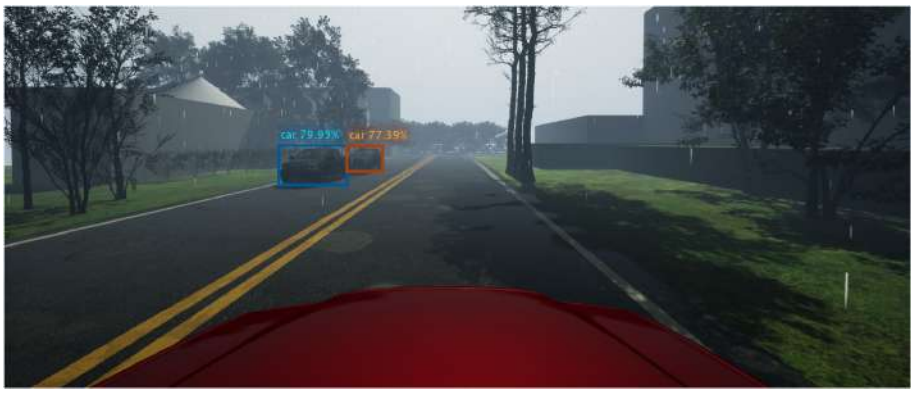


Figure 3.18: Recreated Scene in Unreal Engine 5

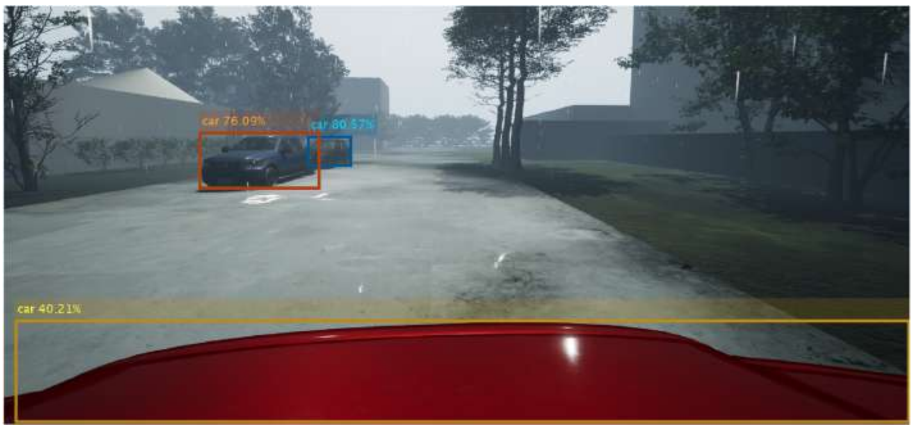


Figure 3.19: After editing the parameters in the scene

Observations and updations

1. **Road Material:** Changing the road material from "Fine American Road" to “Concrete Road” had a significant impact on object detection. This change likely influenced the material’s reflective properties, which can affect how objects are detected through visual recognition systems.

2. **Puddle Layer:** In the original scene (see Figure 3.17), the concrete road was wet, introducing a specific visual quality. To mimic this effect, mesh painting techniques were employed and multiple puddle layers introduced. However, the density of these puddle layers played a crucial role in determining the appearance of wetness. Increasing density created a glossy appearance, and the detection got close to the original scene.

3. **Shadows:** Shadows generated by objects, such as trees, can substantially affect object detection. Longer shadows compared to the original scene can lead to discrepancies in object recognition. Adjusting the position of the Directional Light source helped to shorten these shadows, thereby improving object detection.

4. **Camera Parameters:** In the original scene, the relative positions of detected cars differed from our initial setup (see Figure 3.18). By adjusting camera parameters, such as focal length or field of view, we managed to bring the cars closer to their positions in the original scene. This fine-tuning resulted in a detection outcome that closely matched the original scene, with an accuracy within $\pm 1\%$ (see Figure 3.19).

3.3.6 Examples of the created scenes in Night light:

The manual editing of the scenes in general was made by changing directional light, sky light, objects location, materials parameters, post processing parameters and by adding or removing objects.

Scene 3
The original photo with YOLO detection in figure 3.20 is observed first, noting 7 distinct detections with varying accuracy percentages. This scene is then replicated in Unreal Engine 5.1 as shown in Figure 3.21. During the recreation, several adaptations were made: the unique star-shaped lens flare wasn’t matched since it’s a purchasable item; different road signs were used due to unavailability of the originals; and reproducing the white car on the middle left was challenging because of interior lighting issues. Furthermore, two cars on the right didn’t meet the desired 5% accuracy range, and the driver’s car was undetectable. It’s noteworthy that scenes with more than 4 cars often have accuracy challenges.

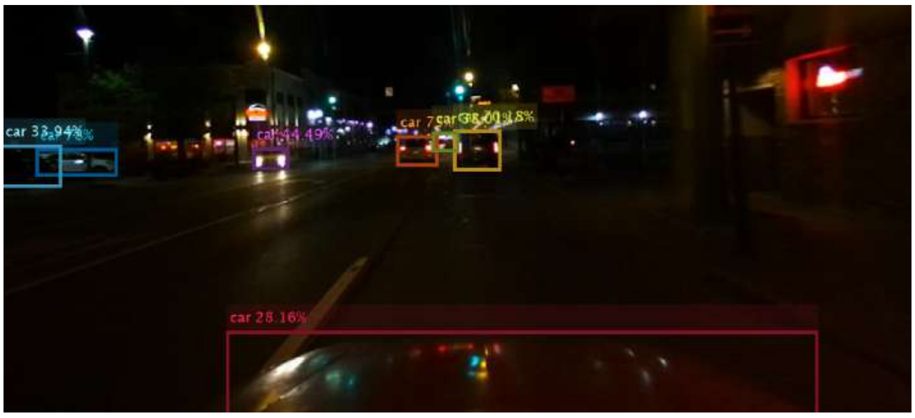


Figure 3.20: Original Image Detection from the Adverse Weather Dataset

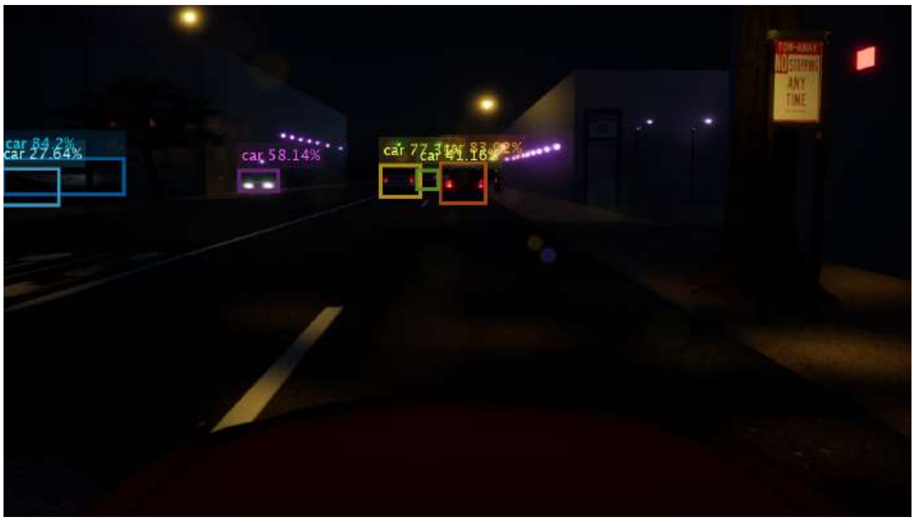


Figure 3.21: Recreated Scene in Unreal Engine 5

Post these observations, the following edits were made to improve the scene:

- Emitting light materials were added to the left building to mimic light emanating from the inside through the windows.
- The intensity of the headlights of the incoming car was adjusted.
- The flare intensity was modified.
- The positions of the two cars on the left were slightly altered.
- The exposure intensity was changed.
- Editing the bloom intensity also helped.

We got the following scene:

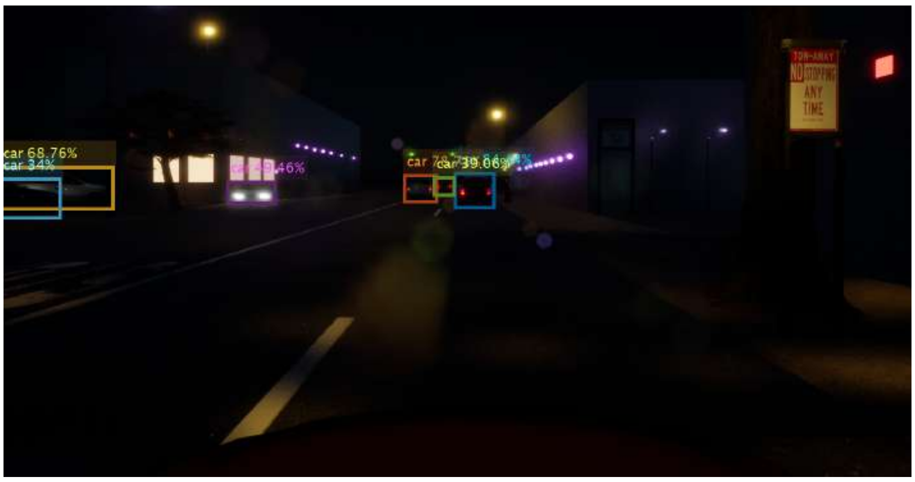


Figure 3.22: After editing the lighting conditions in the scene

After refining the scene with the mentioned edits as can be seen in Figure 3.22 , it's evident that only one car on the right remains outside the desired accuracy range. Additionally, the adjustments to the car in the middle left, specifically its orientation combined with the addition of emissive light windows to the building on the left, have brought it more in line with the original, enhancing the overall authenticity of the recreated scene.
Scene 4
Upon analyzing the original scene utilizing the YOLO detection system in Figure 3.23, we found a total of six distinct detections. When the scene was subsequently recreated and subjected to initial edits as seen in Figure 3.24, a number of differences were evident. Notably, the car on the left failed to achieve the desired 5% accuracy range. Furthermore, the recreated scene's illumination was notably low in contrast to the original image. Anomalies were also identified in the reflection on the primary vehicle, which functioned as the camera's base, and in the brightness of the road lights on the right side
Subsequent refinements to the scene shown in figure 3.25 involved a series of adjustments:

- The alignment of the car on the left remained a challenge, despite modifications to its orientation, luminance, and scale.

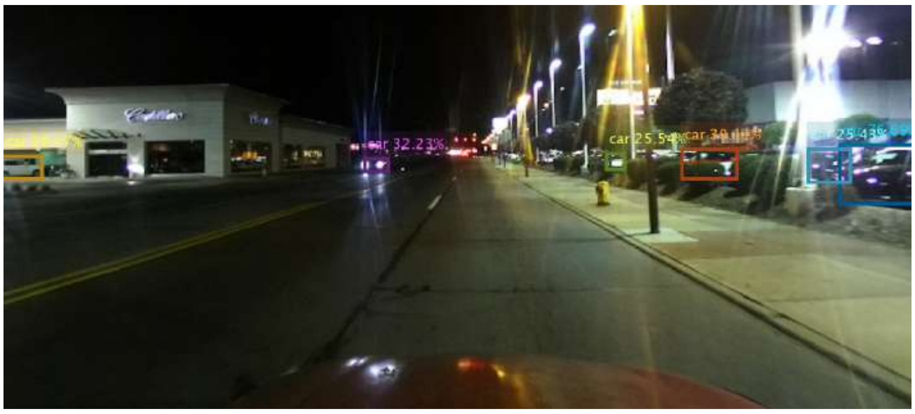


Figure 3.23: Original Image Detection from the Adverse Weather Dataset

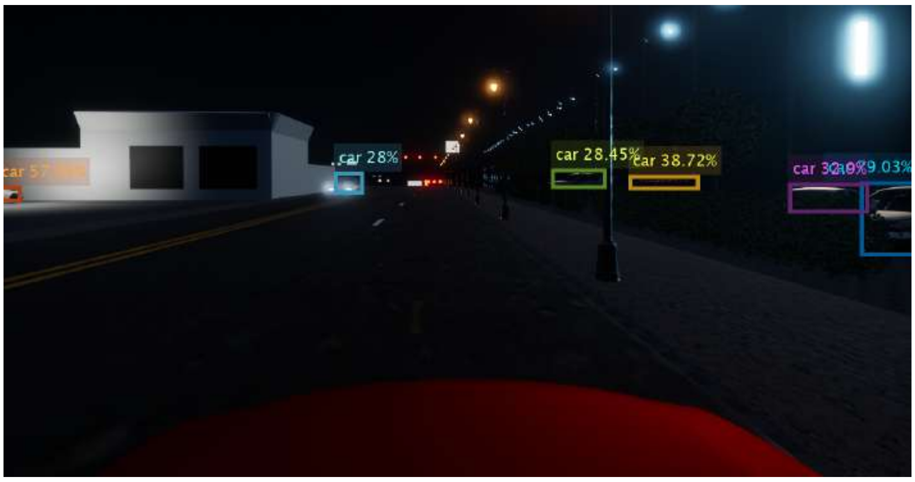


Figure 3.24: Recreated Scene in Unreal Engine 5

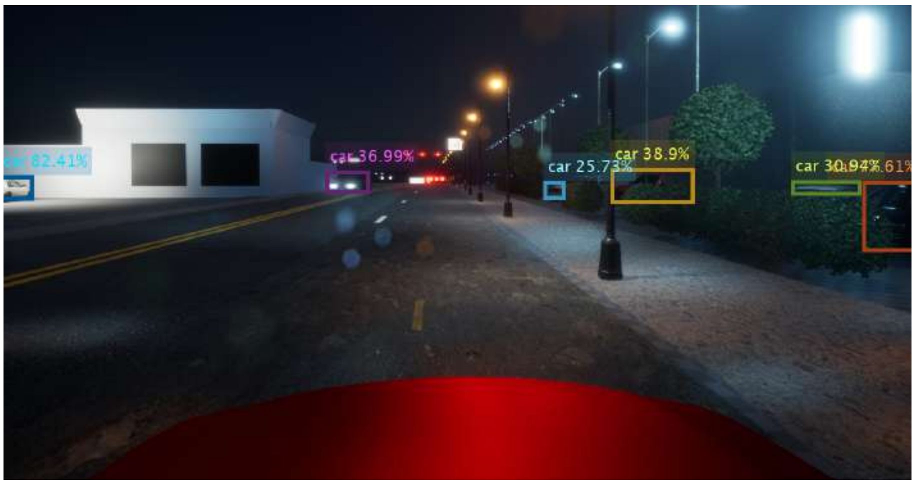


Figure 3.25: After editing the lighting conditions and adding more details

- The intensity of the sky lighting and direction was enhanced to better correspond with the original scene.
- There was an increase in the light flare to better capture the ambiance.
- The intensity of the road lights on the right was increased.
- Adjustments were made to the reflection on the main car, specifically its position.
- The elevation of the cars positioned on the right was changed.

These cumulative modifications significantly improved the fidelity of the recreated scene, moving it closer to the original photograph's representation.

3.4 Limitations of creating the scenes

Creating scenes for automated driving simulations in Unreal Engine and incorporating vehicle and pedestrian detection using models like YOLOv5 offers great potential for testing and validating autonomous systems. However, there are several notable limitations to this approach.

One of the primary challenges is the trade-off between realism and accuracy. While Unreal Engine can generate visually impressive and realistic environments, achieving a perfect match with real-world conditions can be difficult. The accuracy of object detection models like YOLOv5 may be affected by the discrepancies between the simulated and real environments, including factors like lighting, weather, and the behavior of pedestrians and vehicles. Simulated environments may also struggle to capture the full complexity and variability of real-world traffic scenarios. Real-world traffic can be highly unpredictable, featuring diverse road conditions, unexpected events, and intricate interactions between vehicles and pedestrians. Simulations might not fully replicate such variability, posing limitations in the testing and validation of autonomous systems.

Additionally, models trained on simulated data may not generalize well to real-world scenarios. The ability of the YOLOv5 model, for example, to detect vehicles and pedestrians in different lighting conditions, varied road types, and diverse urban environments may be limited. Fine-tuning or transfer learning might be necessary to adapt the model to real-world conditions effectively.

Moreover, running deep learning models like YOLOv5 in real-time within a simulation environment can be computationally demanding. Achieving the necessary frame rates for real-time simulations often requires high-end hardware, which can be costly and limit scalability. Validation presents another set of challenges. Metrics derived from simulations may not always correlate perfectly with real-world performance, making it essential to verify and validate the system against real-world data. This process can be intricate and time-consuming, requiring careful consideration and planning.

Developing and maintaining complex simulations in Unreal Engine can be resource-intensive, both in terms of development time and hardware requirements. Creating large-scale, highly detailed environments with diverse scenarios can demand significant investments in resources.

A clear representation of real-world driving conditions is paramount. However, some challenges have been noted with the original scenes in the engine. Firstly, road directions in many scenes are ambiguously designed and lack clear lane markings and signs, making it difficult for understanding the ending and beginning of the scenes. Secondly, the lighting in these scenes, often not so clear to know the source or approximate how far it is.

3.5 Conclusion and evaluation

In conclusion, this work package delves into the critical areas of Environment Modeling as it pertains to automated driving simulations, focusing on the integration of Unreal Engine and the YOLOv5 algorithm to create realistic scenarios for autonomous vehicle testing and training.

Also, Unreal Engine provides a dynamic and immersive environment for the development and testing of autonomous vehicles. It enables the creation of realistic scenarios, enhancing the realism of simulations. The accuracy and reliability of the YOLOv5 model is also commendable. It effectively detects and tracks both pedestrians and vehicles across various scenarios, not only in the scenes based on the Original Dataset but also within those meticulously recreated in Unreal Engine 5.1. This outcome not only underscores the robustness of the YOLOv5 model but also signifies its adaptability and usability within the dynamic virtual environments essential for automated driving simulations. The successful integration of the YOLOv5 model into Unreal Engine 5.1 hints at its immense utility in conducting automated driving simulations. Such simulations hold the promise of not only enhancing efficiency but also guaranteeing safety and cost-effectiveness in the development of autonomous vehicle technologies.

Furthermore, the strategic combination of environment modeling and YOLOv5 detection affords the creation of a comprehensive testing arena for automated driving systems. This simulated environment not only enables rigorous testing under diverse conditions but also drastically diminishes the dependency on costly physical testing. This dual advantage—enhanced safety and cost-effectiveness—has the potential to reshape the landscape of autonomous vehicle development, making it more accessible and sustainable in the long run.

3.6 Outlook

As we reflect upon environment modeling for automated driving using Unreal Engine and the engine is improving and becoming closer to real life. It's evident that the future is promising. One of the possible improvements is the implementation of mapped memory code. This approach promises a transformative shift in our current methodology by enabling the automatic adjustment of parameters through connecting Unreal Engine and Matlab directly. Instead of the often tedious manual pipeline t, this automated mechanism will streamline processes, ensuring a more efficient and error-free adjustment of environmental conditions and will also save time.

Another significant evolution is the shift from static to dynamic environments. While static models provide a foundational testing bed, the future lies in scenarios that are ever-evolving and can mirror real-world unpredictability. Dynamic environments are not just about changing visual elements; they encompass variations in traffic density, pedestrian behaviors, and infrastructure modifications, representing a true-to-life replication of the daily variances one might encounter on the road.

Moreover, as our understanding of automated driving systems deepens, there's a pressing need to introduce a wider range of weather scenarios and even diving deep into extreme cases. These would encompass situations like downpours, blinding sandstorms, or heavy snowfall. By expanding the range of simulated scenarios, we ensure that automated driving systems are rigorously tested for almost any conceivable situation they might face.

Disclaimer:

For the purpose of enhancing linguistic clarity, coherence, grammar and overall structure, this chapter has been reviewed and refined using ChatGPT, an artificial intelligence language model developed by OpenAI. While ChatGPT can generate and improve human-like text, it doesn't inherently validate the accuracy or factual content of the report. For the text in the report, we wrote the content based on documentation and used ChatGPT to enhance this text to a more technically sound language.

4 Octomap for environment modelling of cobots

The primary focus of workpackage three was to conduct experiments and tests related to 3D mapping and occupancy modelling using a UR10 robot and a Microsoft Azure Kinect camera. The fundamental idea was to use and modify the ROS package *Octomap*. The *Octomap* package subscribes to the Pointcloud data generated by the depth camera. This data was then used to construct the Octomap, an occupancy grid of voxels, to map and model objects of interest.

Octomaps offer valuable capabilities in representing the environment as a probabilistic occupancy map. It has important features in object recognition, collision avoidance, and path planning. The central objective of this workpackage is to assess and enhance the efficiency of object modelling through the use of Octomap mapping techniques. This involves capturing multiple viewpoints of the object using the UR10 robot.

To provide a clear structure to this chapter, it is divided into following sections. Firstly, there will be an introduction into Octomaps, where their advantages and key features are discussed. Following this, the project setup and the Octomap integration will be discussed, with a special focus on optimizing the utilization of Octomaps and exploring various approaches to achieve this. Additionally, a brief overview about next best view planners will be provided, which play a role in guiding the robot's movements for effective mapping. Lastly, the chapter concludes with an evaluation of the project's outcomes, providing insights into the success and challenges encountered during the project.

4.1 Octomap

In this initial section, a brief introduction to the *Octomap* package will be given. The advantages of the *Octomap* package will be depicted, and different approaches to model a 3D environment will be discussed. Section 4.1.2 will be talking about the compression method and structure of Octrees. Octomaps make use of occupancy estimation to model the objects in the environment, which will be talked about in section 4.1.3.

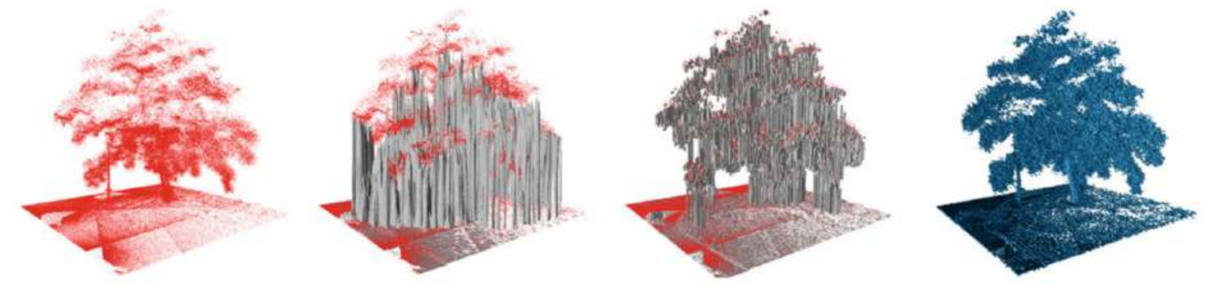
4.1.1 What is Octomap ?

Octomap is an open source library, that is used to generate 3D occupancy grid maps from sensor data. The basis of this package is the research of (Hornung et al. 2013). The Octomap is a 3D occupancy map that is using a probabilistic occupancy estimation approach. It is based on an Octree data structure, used for higher flexibility in the mapped area, as the resolution of the Octree is continuously adapted to the environment. An Octree is a tree data structure that is divided into eight octants or sub-volumes (Wilhelms and Van Gelder 1992, pp. 206–207). Each node in the tree has eight children, that are either empty or full. Therefore, a node in the tree represents a volume in 3D space, also called voxel. The *Octomap* package uses a probabilistic estimation to decide if a voxel is occupied, free or unknown.

Nowadays, different approaches are used to map environments as a 3D model. An often used modelling approach is working with Pointclouds. A Pointcloud is a set of points in a 3D space, where every point is defined by its coordinates $p = [x, y, z]^T$. A major drawback of this method, is that Pointclouds do not model free or unknown space (Hornung et al. 2013). The difference between free space and unknown space is not taken into account. In addition to this, Pointclouds store a large amount of data. While Pointclouds provide direct spatial information, they can be sparse and they require additional processing for efficient object recognition. Octomap mitigates sparsity by maintaining a structured volumetric representation and therefore tends to have some good attributes for object recognition.

Other common approaches that are used for 3D modelling are elevation maps and multi-level surface maps. Both maps do not model free space or unknown space, whereas Octomap explicitly models free space (Hornung et al. 2013). In Figure 4.1 a tree was modeled with different approaches while using a laser range sensor. In Figure 4.1a the tree is modeled with a Pointcloud, in Figure4.1b with an elevation map, in Figure4.1c with a multi-level surface map and in Figure 4.1d with an Octomap. The Pointcloud representation as in Figure 4.1a only models the visible surface of the tree, whereas the Octree in Figure4.1d also models the interior of the tree. The elevation map and the multi-level surface map are more or less optimized for relief mapping and not for object modelling and object detection. Elevation maps lack the ability to model vertical surfaces and also different levels of elevation. Elevation maps only store the height of the highest surface in a specific area. Multi-level surface maps are able to model vertical surfaces, but they are more suitable for modelling large scale environments (Triebel, Pfaff, and Burgard 2006). Therefore Octomap representation is more suitable for object modelling and detection. In Figure 4.1 it is clearly visible that the Octomap representation is the most detailed representation of the tree.

A node is initialized in the Octree if the space is occupied. Any uninitialized nodes could be represented as free space or as unknown space. To bypass this uncertainty, free volumes are explicitly modeled between the sensor and the endpoint.



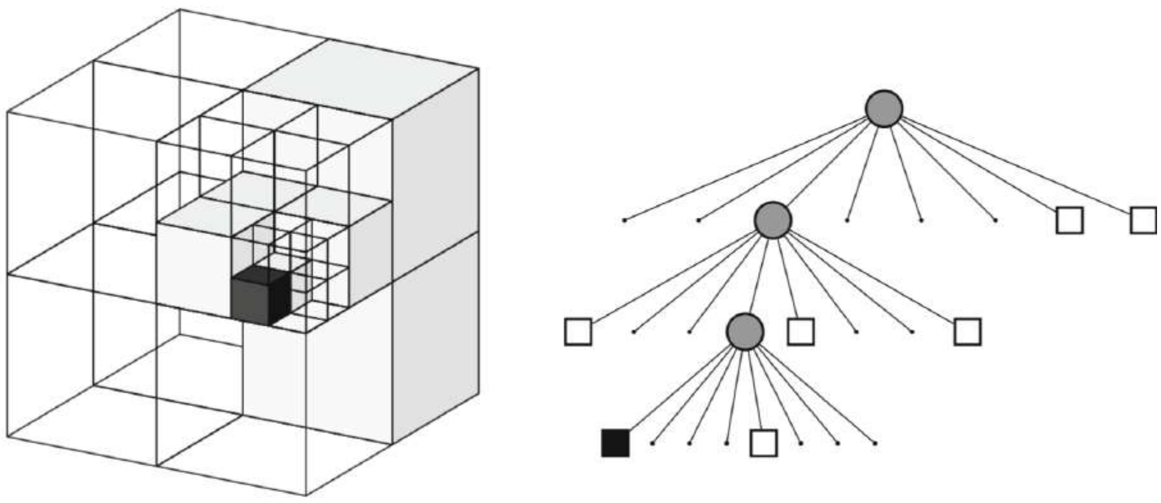
(a) Pointcloud representation (b) Elevation map representation (c) Multi-level surface map representation (d) Octomap representation

Figure 4.1: 3D representations of a tree scanned with a laser range sensor (Hornung et al. 2013)

4.1.2 Octomap compression method and structure

The Octree representation and usage as in (Hornung et al. 2013), cuts off the Octree at a certain depth, when all children nodes have the same state. This is a compression method that reduces memory usage, by combining map volumes. The compression method reduces the memory usage of the Octree map. Children of a common node, that have the same occupancy state, are pruned. Should the state of a node change, then it is expanded again. Maximal spatial extend of the Octree is limited by the maximum depth of the Octree. The maximum depth of the Octree is set to 16 (Hornung et al. 2013). The resolution of the Octree is set by defining the minimum size of a voxel.

In Figure 4.2 an Octree structure is represented. The volumetric model of the Octree is displayed in Figure4.2a and the corresponding tree representation is displayed in the Figure4.2b. The nodes in the shaded white cells are free and the nodes in the black cells are occupied. In both representations, there are some unknown cells that are neither marked free or occupied. In Figure 4.2 the compression method is clearly visible. Volumetric space that doesn't need to be mapped is kept at the highest possible level of the Octree. On the other side, volumetric space that needs to be modeled is mapped on a lower level of the Octree. In this figure, the occupied cells are mapped on a lower level of the Octree.



(a) Volumetric representation of an Octree (b) Tree representation of an Octree

Figure 4.2: Example of an Octree storing free (shaded white) and occupied (black) cells (Hornung et al. 2013)

4.1.3 Occupancy probability estimation

The *Octomap* package is using an occupancy estimation to model the objects in the environment. Octomap ensures updatability and copes with sensor noise by using a probabilistic occupancy estimation. A threshold on the occupancy probability is used to determine if a node is occupied or free. The probabilities of the children nodes are taken into account, when calculating the probability of a node. One major strength of Octomap is the handling of occlusion and estimation of the size of an object. The *Octomap* package calculates a probability of occupancy for each node and updates each probability with every new measurement (Hornung et al. 2013). The formula for the occupancy probability is displayed in (4.1.1). It depends on previous measurements, the previous probability of occupancy $P(n|z_{1:t-1})$ and a prior probability $P(n)$. $P(n)$ is often set to 0.5, because it is assumed that the environment is equally likely to be occupied or free. The probability of occupancy $P(n|z_{1:t})$ is calculated from the sensor measurement z_t. By simplifying (4.1.2) as in (4.1.3), the probability of occupancy can be converted in a log odds representation. The log odds probability representation is favorable, because it can be added and subtracted and is therefore more efficient to calculate (Hornung et al. 2013).

$$P\left(n|z_{1:t}\right) = \left[1 + \frac{1 - P\left(n|z_t\right)}{P\left(n|z_t\right)} \frac{1 - P\left(n|z_{1:t-1}\right)}{P\left(n|z_{1:t-1}\right)} \frac{P\left(n\right)}{1 - P\left(n\right)}\right]^{-1} \tag{4.1.1}$$

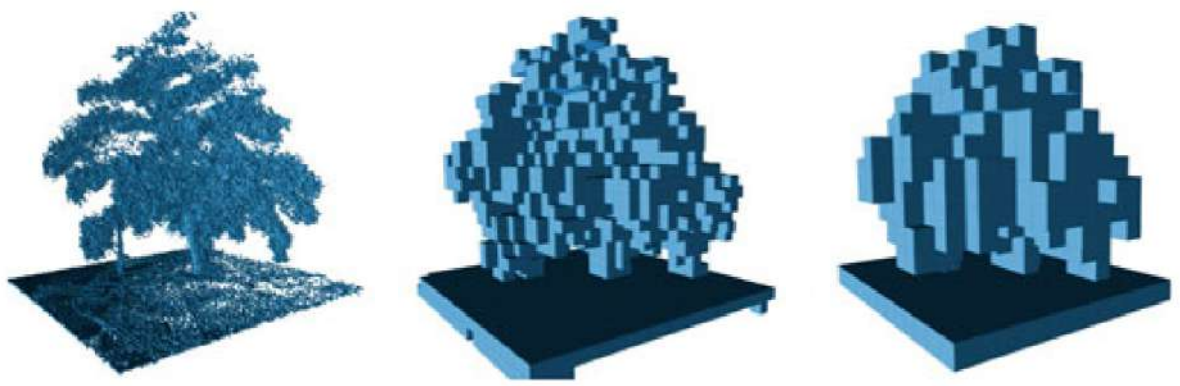
$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t) \tag{4.1.2}$$

$$L(n|z_t) = \log\left(\frac{P(n|z_t)}{1 - P(n|z_t)}\right) \tag{4.1.3}$$

To change the state of a voxel, the new state needs to be observed at least as long as the old state was observed. To allow faster voxel manipulation, a boundary was set to enable flexible changes in the occupancy state of a voxel. Depending on the time of visual, temporary appearing obstacles are not permanently modeled in the Octomap. As an example, a person is walking through the environment, then the *Octomap* package will not model the person, as long as the person is not standing still. As soon as the package recognizes some static environment, the environment will be modeled in the Octomap. The package will not model any movement. On the other hand, static obstacles are permanently modeled in the Octomap. Objects that exceed a certain time limit and have static criteria are modeled in the map. The *Octomap* package creates an Octree map for a static environment.

The Azure Kinect camera used in the project can be moved around to model the environment. But the environment needs to be static and the camera position needs to be known at all times. The *Octomap* package is not able to model a dynamic environment or to create a dynamically changing Octomap. Rotating or moving objects while modelling the environment, leads to a blurred and disfigured Octomap. In the Section 4.2.3 the different approaches to create a dynamic Octomap are depicted.

In Figure 4.3 a tree is modeled with different resolutions. The tree is modelled with a resolution of 0.08m in Figure 4.3a, 0.64m in Figure 4.3b and 1.28m in Figure 4.3c (Hornung et al. 2013, p. 192). As seen in the figure the resolution of the Octomap has a huge impact on the quality of the Octomap. During the project, a challenge was to find a good balance between the resolution of the Octomap and the computational effort, having an impact on the frequency of the Octomap.



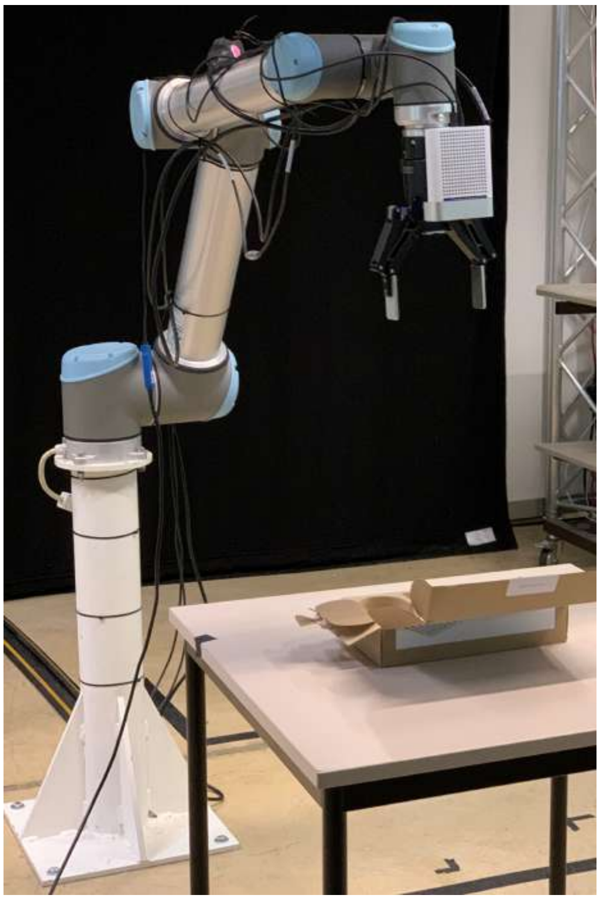
(a) Octree with a resolution of 0.08m

(b) Octree with a resolution of 0.64m

(c) Octree with a resolution of 1.28m

Figure 4.3: Example of a tree model, displayed with different resolutions (Hornung et al. 2013)

4.2 Integration of the Octomap package


(a) UR10 robot with the azure kinect camera mounted on the end effector

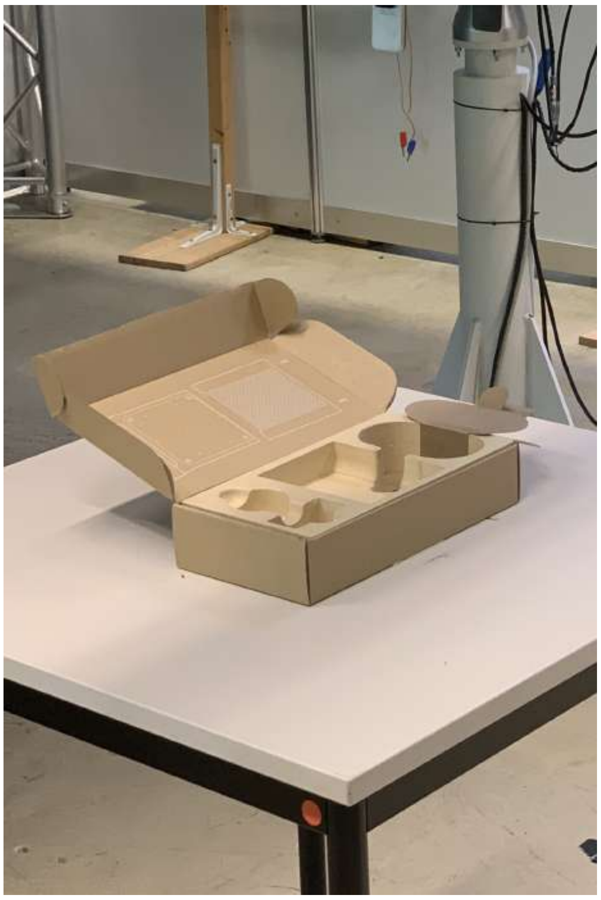

(b) Azure kinect camera box (object)

Figure 4.4: Robot and camera setup

The Azure Kinect camera was used to record a Pointcloud of the environment. After remapping the *pointcloud2* topic to the Octomap server, an Octomap of the environment was created. The *octomap_server* package subscribes to the *pointcloud2* topic and publishes the Octomap on different topics. During the project a UR10 robot was used to move the camera around the object (*Universal Robots* 2023). The camera was fixed on the end effector of the robot. The setup is depicted in Figure 4.4a. In Figure 4.4b the Azure Kinect camera box is displayed, which will be later on modeled with the *Octomap* package. In Figure 4.5 a simulation in RVIZ of the robot with the Octomap and the Pointcloud is displayed. The azure kinect camera box was modeled with a resolution of 0.01m.

The Octomap is used for different purposes in this project. The first purpose is to create a static map of an object, that has to be analyzed and modeled by the robot. The second purpose is to create a dynamic map of the environment, that is used by the robot to navigate in the environment and to avoid obstacles. The actual Octomap code was changed in order to adapt the Octomap to the needs of the project.

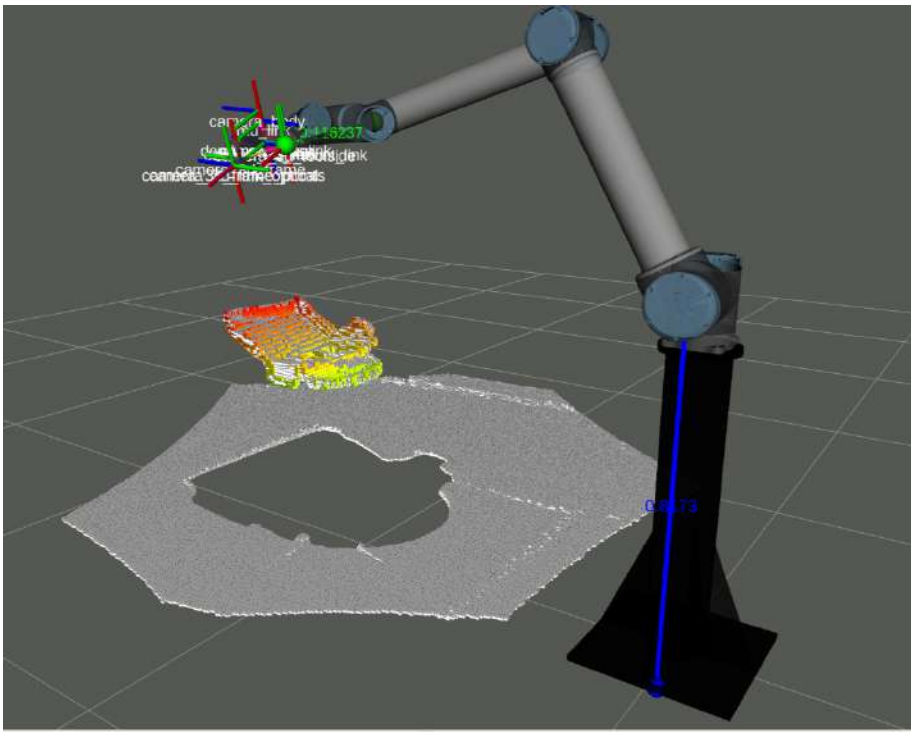


Figure 4.5: RVIZ simulation of the Robot with the Octomap and the pointcloud

In the next section, a brief introduction to the Azure Kinect camera will be given. Followed by a discussion about Octomap optimization and depicting various approaches to creating dynamic Octomaps.

4.2.1 Azure Kinect Camera

Kinect contains a depth sensor, spatial microphone array, a video camera, and orientation sensor as an all in-one small device with multiple modes, options, and software development kits (SDKs) making it easier to integrate its capabilities into different applications.

One of the features is its Depth camera sensing capabilities. It employs a time-of-flight sensor that measures the distance in the camera's field of view. This depth information is crucial for creating detailed 3D representation of the environment. The camera is capable of real-time data processing, which is advantageous for applications requiring quick decision-making and interaction with the environment, such as robot navigation and interaction with objects.

The Sensor SDK has the following features that work once installed and run on the Azure Kinect DK:

- Depth camera access and mode control (a passive IR mode, plus wide and narrow field-of-view depth modes)
- RGB camera access and mode control (for example, exposure and white balance)

- Motion sensor (gyroscope and accelerometer) access
- Synchronized Depth-RGB camera streaming with configurable delay between cameras
- External device synchronization control with configurable delay offset between devices
- Camera frame meta-data access for image resolution, timestamp, etc
- Device calibration data access

In this project, we have used Azure Kinect camera ROS package provided by Microsoft for development purpose. The camera's main use was to generate Pointcloud data using its depth sensor. The depth sensor was used in Narrow Field of view (NFOV) mode. The mode has a minimum operating range of 0.5m with 75°x65° Field of Interest (FoI). In this mode, the sensor was able to generate Pointcloud data at 30 frames per second (FPS).

4.2.2 Octomap optimization

In this phase of the project, different parameters of the Octomap were adjusted to improve the performance of Octomap. Various parameter changes were tested to observe their impacts. The primary objective was to find a balance between the resolution and the rate of the Octomap creation. Lowering the Octomap 's resolution resulted in an enhanced frequency of Octomap generation. But lowering the resolution also resulted in a loss of detail in the Octomap.

In numerous scenarios, only a portion of the environment is relevant. As a feature of the package, the Octomap creation can be restricted to a defined range. The maximum sensor range, which can be configured in the launch file, represents the farthest distance from the sensor to its surrounding environment. Decreasing the sensor range results in a reduction of the Pointcloud data processed by the Octomap. The resolution of the Pointcloud is higher than required, a voxel grid filter was applied to downsample the Pointcloud. The Pointcloud was downsampled with the voxel grid filter from the *perception_pcl* package, as part of the *pcl_ros* package (*perception_pcl* 2023). *pcl_ros* and *perception_pcl* are ROS packages, that enable different Pointcloud processing functions for the Pointcloud Library.

In dynamic scenarios, achieving a high frequency of Octomap creation is of great importance. Rapid Octomap generation is essential for timely responses to environmental changes. To measure the impact of parameter changes, two testing methods were adapted: (i) a rosbag file was recorded for simulation and (ii) real environment using a UR10 robot. The different parameters changed were resolution of Octomap r_{oct}, sensor range of Octomap $s_r$and pcl filter voxel size vf_s. The frequency of Octomap creation f_{oct} was then evaluated.

Initially, experiments were conducted on method (i). The r_{oct} was alternated between 0.01m and 0.07m. The s_r was changed between 1.5m and 2.0m. The category vf_s represents the chosen size for the individual cubic voxels in a voxel grid filter, which affects how finely or coarsely the data is organized and filtered. The voxel leaf refers to the size of the cubic voxels used to discretize the Pointcloud. The vf_s was altered between 0.01m to 0.03m and a test without using pcl voxel filter was conducted. At this point, it is important to note that the vf_s needs to be smaller or equal to the r_{oct}. Otherwise, the occupied space will not be modeled correctly and have gaps of free space. Further evaluations are in Section 4.4.

During the testing method (i), some outliers became noticeable in the Pointcloud. The rosbag was recorded in a room with numerous windows and ample light, resulting in substantial noise in the Pointcloud. In testing method (ii), the surroundings were less noisy and the outliers were less prominent. Therefore, in method (i), an additional filter was applied to remove the outliers. The outlier removal was implemented with the help of the *perception_pcl* package and by using the function *pcl::RadiusOutlierRemoval* (*perception_pcl* 2023). This function removes outliers from a Pointcloud dataset using a radius search. The radius search considers the number of neighbors found within a certain radius. In this case, the number of neighbors are compared to a threshold value. If the number of neighbors is below the threshold, the point is considered an outlier and removed from the Pointcloud. This filter was applied after the voxel grid filter. The outlier removal had a positive impact on the frequency of the Octomap creation in method (i). At this point, it is important to note that the outlier removal was not applied in method (ii). Since, the real testing environment was less noisy. When working with a higher resolution, the accuracy of the outlier removal filter needed to be increased. In this case, an additional overhead can be observed when using outlier filter. Depending on the environment, the outlier removal can have a positive or negative impact on the frequency of the Octomap creation and needs to be tested when changing environments.

Further, to evaluate the previously obtained results with method (ii), a series of tests were performed. These tests took place in a less noisy environment and characterized by improved lighting conditions. This reduction in noise was achieved through the use of an enclosed space and the deployment of black curtains to maintain consistent lighting conditions. Figure 4.4a provides an illustration of the experimental setup. During these tests, the azure kinect camera box was modeled at various resolutions. In Figure 4.4b the azure kinect camera box is displayed. The UR10 robot moved the camera to an elevated position, directly overlooking the azure kinect camera box and due to that the Octomap was set to dynamic mode.

In method(ii), the sensor range was reduced to 1.1 meters, effectively limiting the data processed by the Octomap to the relevant area. The outcomes of these tests and resulting models of the Azure Kinect camera box are also documented further in evaluation section 4.4.

4.2.3 Static Octomap and dynamic Octomap creation

As explained before, the goal of using the Octomap was creating a precise model of an object and also being able to avoid obstacles in the environment, while moving the end effector of the robot. While the end effector is reaching for a feasible position, the Octomap can be used in the dynamic mode. When the end effector gets to the desired position, the Octomap can be switched to the static mode to create a precise model of the object. In this section different approaches are taken into account to create a dynamic Octomap.

To achieve a dynamic Octomap , the code was adjusted to remove nodes from the Octree that no longer appear in the Pointcloud. The Octomap recognizes and stores incoming nodes representing obstacles but doesn't remove older obstacle nodes. It only eliminates nodes that temporarily vanish but no longer form obstacles. This method isn't suitable for dynamic environments where obstacles move, and the robot must react to environmental changes.

During the project, different approaches were tested to create a dynamic Octomap. The first approach was to create a static Octomap and to regularly reset the Octomap with the service call reset. When executing the service call reset, the Octomap is completely deleted and rebuild. The second approach was to use the service call clearBBX to regularly clear the Octomap. With the service call clearBBX, a bounding box can be defined, which is then cleared from the Octomap. The bounding box sets all occupied nodes within the bounding box to free. But both approaches were not suitable for the project, as the Octomap server took too long to clear the Octomap. Working with service calls can be very time consuming and can block different resources.

The Octomap's code underwent a modification where previously stored nodes are removed from the Octree if they no longer appear in the current Pointcloud data. This process involves accessing Octree nodes through an Octree node iterator. To update the occupancy status of these nodes after the removal of the outdated ones, the *updateNode()* function was employed. This adjustment ensures that the Octomap remains up-to-date and accurately reflects the evolving environment captured by the Pointcloud. A flowchart of the dynamic Octomap creation is depicted in Figure 4.6 to illustrate the process.

A node was implemented to facilitate the access to both the static and dynamic Octomap. This node enables the transition between the two states through a service call and can also pause the Octomap creation during a camera movement in the static mode. To create a precise model of an object while using different viewpoints, the Octomap can be switched to the static mode. During the movement of the end effector, the Octomap can be paused through another service call, to avoid modelling the surrounding environment.

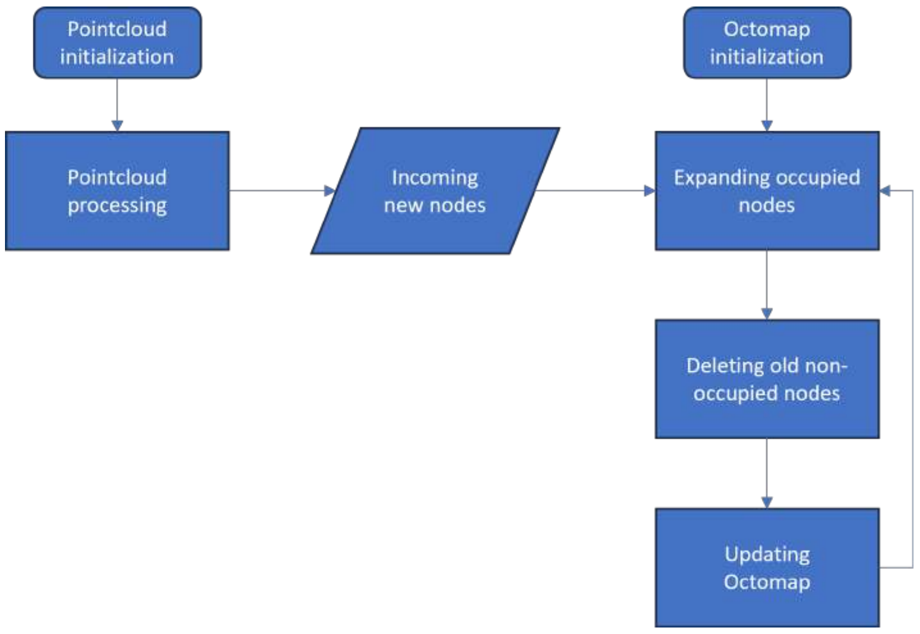


Figure 4.6: Flowchart of the dynamic Octomap creation

4.3 Introduction into NBV Planners

Capturing and building a complete 3D map of an object from one viewpoint does not provide sufficient information. For autonomous exploration, the robot should move to different viewpoints for a complete map. The selection of these viewpoints manually requires a lot of time and effort. To decrease the computational efforts, algorithms such as Next Best View (NBV) planners can be used.

4.3.1 Definition

The NBV algorithms are used for autonomous exploration of unknown 3D space. They refer to intelligent selection of how and where the robot should capture the next images. The goal is to increase the information gain and reduce the time & resources required to obtain a scene reconstruction of sufficient quality. The environment is mapped in an occupancy map constructed from depth images input. In the resulting known free space, a finite iteration random tree is grown using a suitable tree construction algorithm (e.g., RRT, RRT*) (Bircher et al. 2016) and each branch is evaluated for unmapped space that can be mapped. The first edge of the best branch is then executed. Any partial view of an object is required to begin re-construction. The algorithm iteratively selects a NBV based on the current model of the object and attempts to navigate to that viewpoint while continuously collecting new sensor measurements to improve the object reconstruction.

This process is repeated until object reconstruction is complete. The NBV search space is dynamically adjusted as the object is reconstructed and is permitted to include unknown space. Three types of applications that make use of NBV algorithms are detailed 3D model reconstruction, environment exploration and mapping, and inspection tasks where basic geometry of the object is known a prior.

The NBV planner works on Environmental analysis and Model analysis as depicted in Figure 4.7. The Environment analysis is mainly on proper visibility of object. The analysis of the environment reasons about the visibility of an object and is carried out based on a set of navigable poses (Mcgreavy, Kunze, and Hawes 2016). Poses in suitable areas are then carried on into the model analysis (B), in which the visibility of object features is evaluated. Conceptually, the resulting areas from (A) and (B) are combined to find the next best view (C).

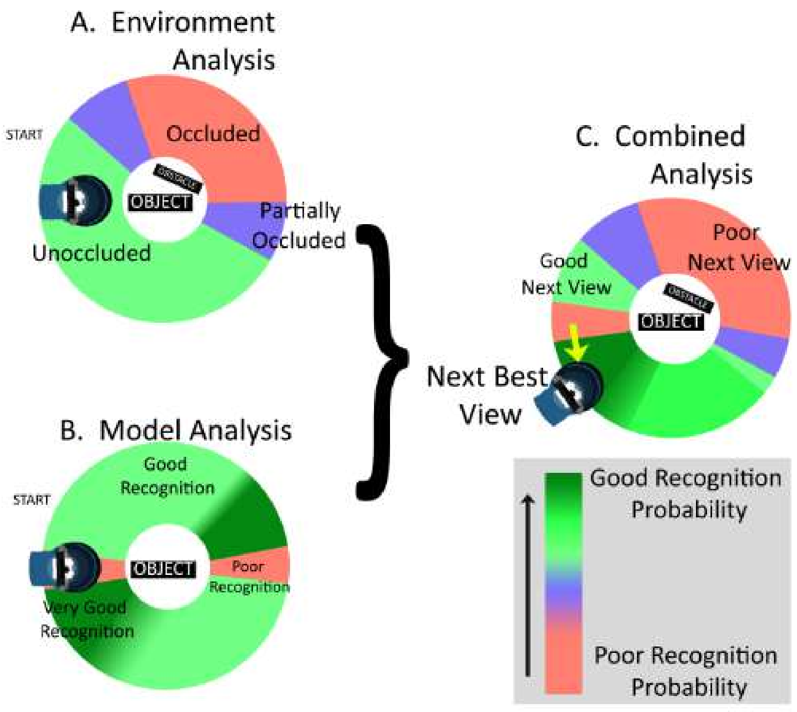


Figure 4.7: Analysis overview of NBV planner (Mcgreavy, Kunze, and Hawes 2016)

4.3.2 Description of the basic next best view package

The ROS package *basic_next_best_view* is a basic next best view planner which can be implemented to get NBV poses as candidates of viewpoints. (Aleotti et al. 2014). The action client server has been developed which returns the requested NBV poses on calling the action. Before calling the action, we need to publish point of interest and a occupancy cloud on the topic which the package subscribes. The point of interest here is the position of the object in the world frame. The Occupancy cloud is the Pointcloud message (sensor_msgs) of XYZI type.

After publishing the required message on topics, the algorithm starts building a virtual sphere considering the point of interest as a center point. On calling the action, it will

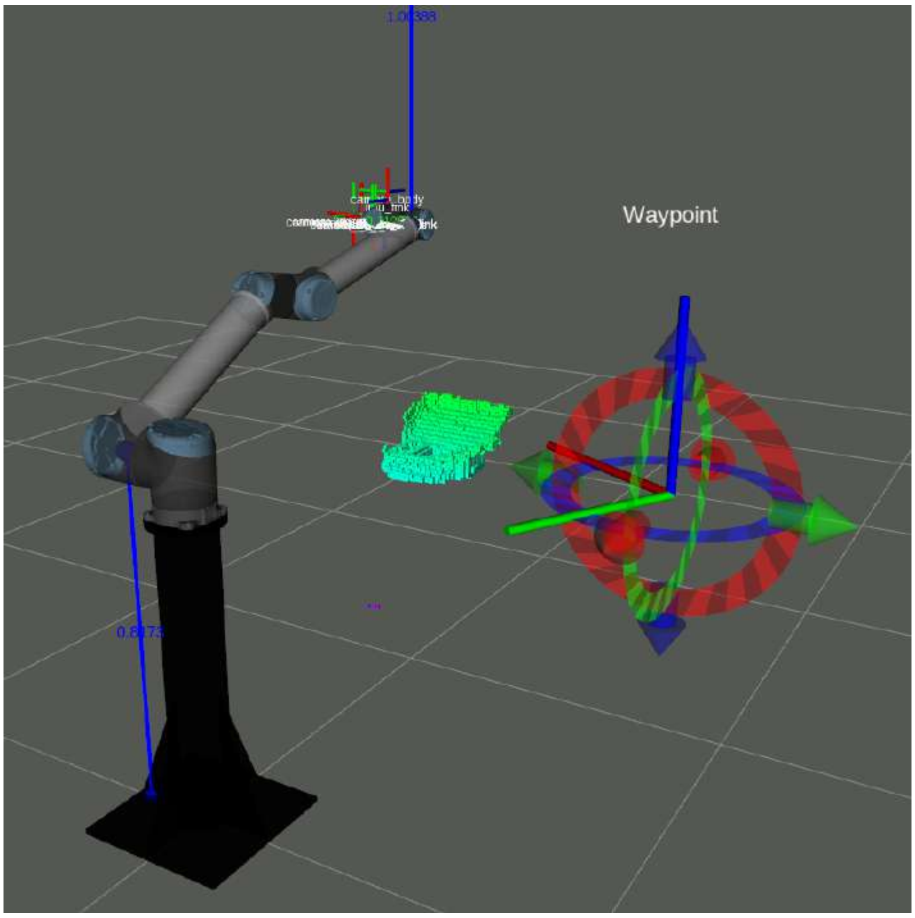


Figure 4.8: Robot with a next best view waypoint and the Octomap of the azure kinect camera box in RVIZ

return an array of poses, ordered from the fittest (at index 0), based on the number of unknown and known voxels visible from that pose, depending on the metric. The metric is chosen by tuning several parameters within the package. In Figure 4.8, the robot is simulated in RVIZ and a next best view waypoint and the Octomap of the azure kinect camera box. The next best view waypoint is displayed as a sphere and was calculated with the basic next best view package.

4.4 Evaluation

Multiple experiments were conducted utilizing the UR10 Robot on which the Microsoft Azure camera was mounted and *Octomap* ROS package was employed to generate occupancy map. The system with specifications Intel Core i5 – 6500 CPU, memory capacity of 32GB and NVIDEA GeForce GTX 1660 was used. While processing the Octomaps, it was noted that multiple cores were running at 100%. The trials were conducted in the robot cell of the Institute of Robot Research, TU Dortmund which had a good amount of natural lighting. The two positions and orientations of the camera considered were in the direction of gripper (end effector link) and with a shift of –90° in X direction. With the initial position, the gripper was obstructing the view which generated occupancy map. This was later evaluated and the change in camera orientation and transformation in frames were made.

To evaluate the performance of our system, a rosbag (i) was recorded and we defined some metrics relevant to the experiment such as frequency of Octomaps, accuracy and precision of occupancy map. The rosbag captured data from the Azure Kinect camera while introducing controlled camera movement and simulating a dynamic situation with a person walking through the scene. This method (i) was then replayed to test the impact of parameter changes on the frequency of Octomap creation. The tests were conducted with dynamic Octomap , that was modified as described in section 4.2.3. These metrics were observed after changing various parameters like resolution & sensor range of Octomap's, pcl filter voxel size to find an improved solution. With decreasing resolution size, the object was modelled with improved accuracy and precision, but the map was updating slowly i.e., the frequency of Octomap was low. With tuning, we could find some values of parameters to exhibit improved metrics. The results of the parameter changes are listed in the Table 4.1.

The frequency of Octomap creation was measured by subscribing to the Octomap topic and counting the number of messages received. The results listed in the Table 4.1 cannot be generalized because the results are dependent on the environment and the movement of the camera. The amount of light and the distance to objects affect the frequency of the Octomap creation. Additionally, when there is no clear background, it can introduce more noise into the Pointcloud, which can reduce the Octomap creation frequency.

The Pointcloud downsampling has a positive impact on the frequency of the Octomap creation. The experiment no. (5) shown in Table 4.1 was conducted without the pcl filter and showed a much lower frequency compared. As represented in Table 4.1, experiment no. (4) and (5), the frequency of Octomap creation has significant impact with and without the pcl filter.

Sr. no.	Resolution $[r_{oct}]$ (m)	Sensor range $[s_r]$ (m)	Voxel filter leaf size $[vf_s]$ (m)	Frequency $[f_{oct}]$ (Hz)
1	0,01	1.5	0,01	0,7-1
2	0,03	2,0	0,025	6-7
3	0,03	1,8	0,025	7-8
4	0,05	2,0	0,025	12
5	0,05	2,0	not used	1
6	0,07	2,0	0,03	18-20

Table 4.1: Frequency of the Octomap creation with different parameters while using a rosbag recording

Subsequently, experiments on the real environment using UR10 robot (ii) were conducted. The outcomes are presented in the Table 4.2. Notably, there is an overall decrease in the frequency of Octomap creation compared to the results obtained from the method (i). The method (i) had a much lower overhead. This disparity is primarily due to the higher computational load during real-world testing, resulting from different processes running on the robot. The UR10 robot, in particular, runs multiple background nodes and has a larger volume of published and subscribed topics, contributing to this increased overhead.

Sr. no.	Resolution $[r_{oct}]$ (m)	Sensor range $[s_r]$ (m)	Voxel filter leaf size $[vf_s]$ (m)	Frequency $[f_{oct}]$ (Hz)
1	0,01	1,1	0,01	1
2	0,03	1,1	0,02	3
3	0,05	1,1	0,04	5
4	0,07	1,1	0,06	6

Table 4.2: Frequency of the Octomap creation with different parameters in the real environment

Figure 4.9 represents the modelling of real environment using Azure Kinect camera box. The modelling was done with several resolutions (r_{oct}) presented in Table 4.2. It is clearly visible that as the resolution is changed, there is noticeable loss of detail in the Octomap. A resolution of 0.01 meters provides the most faithful representation of the Azure Kinect camera box, closely resembling the actual object as shown in Figure 4.4b. A resolution of 0.07 meters, on the other hand, provides a much coarser representation of the object. Depending on the application, a lower resolution may be sufficient. But for precise modelling of the environment, a higher resolution in the range of 0.01 to 0.03m is recommended. Especially for object detection and object recognition, a higher resolution is needed. When the robot needs to grab an object, a higher resolution is needed to detect the object and to calculate the position and dimensions of the object. On the other hand, a resolution of 0.07 meters is sufficient for basic modelling of the environment, for navigation and especially for obstacle avoidance. A frequency of 6 hz can be sufficient for obstacle avoidance, depending on the speed of the robot. As mentioned earlier, the vf_s should be smaller or equal to r_{oct} for better modelling.

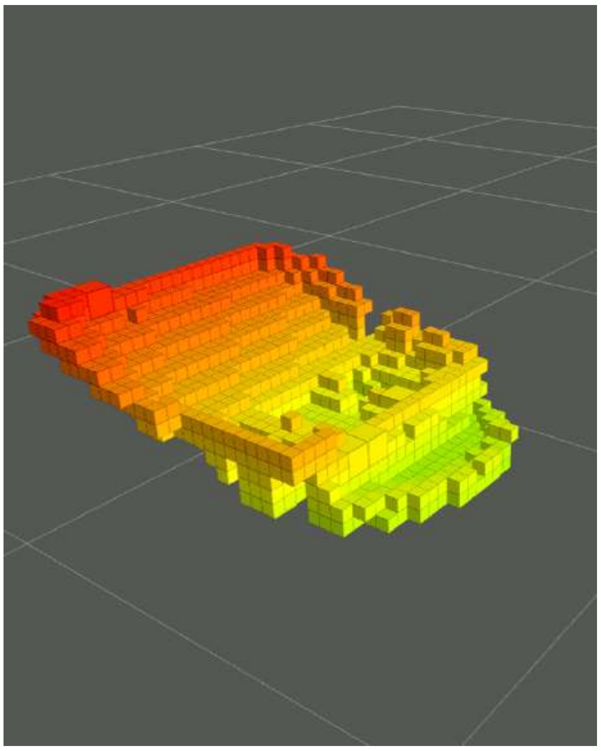
To overcome the problem with dynamic obstacles, a snippet of code was added and modified in the package. The occupancy map could be then operated in dynamic or static mode, depending upon the use case. The dynamic mode deleted and updated the Octomap once the environment was changed while the static was able to preserve the Octomap of the previous and updated environment. In Figure 4.10, the Octomap created while running the rosbag recording is displayed in the static and dynamic mode. In the static mode, the Octomap is only adding new nodes. The static Octomap has great difficulties representing dynamic objects. While in the dynamic mode, the Octomap is deleting and updating the Octomap continuously. In Figure 4.10c, the dynamic Octomap is overlapped by the pointcloud. The contours are still visible in the dynamic Octomap.

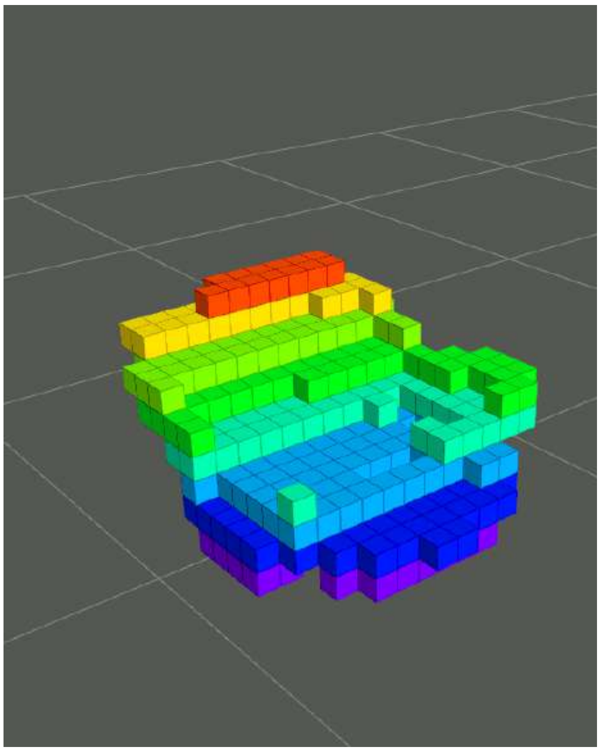
While the system was performing good, it sometimes encountered difficulties exhibiting certain robot positions for some viewpoints of the NBV. This problem could arise with the joint limitations and self-collision in robotic manipulator. Another thing to consider was proper calibration, mounting and transformation of camera frame on the robot.

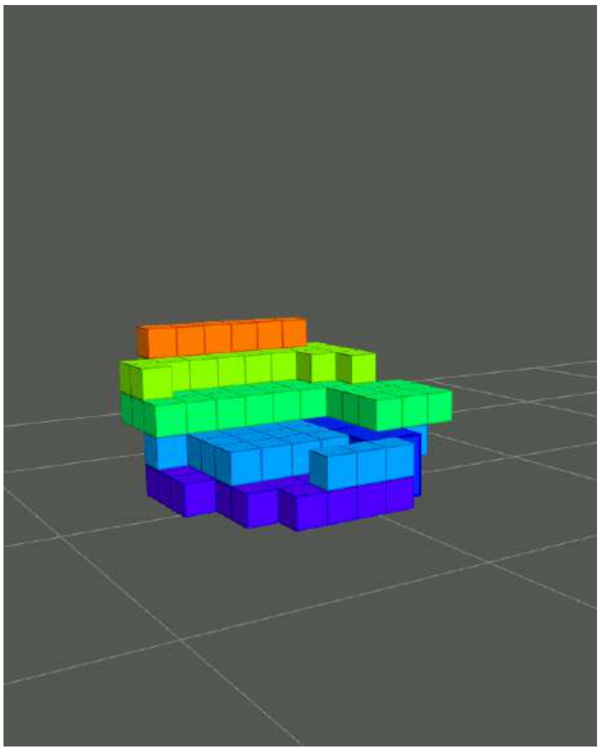
When reducing the resolution of the Octomap , it becomes less accurate. It's crucial to see if it can still represent objects reliably, especially for safety reasons. The occupied space must not shrink when using lower-resolution Octomap's. Otherwise this could lead to collisions with objects. To ensure safety while using lower-resolution Octomaps and moving the robot, the occupied space in the lower-resolution Octomap must overlap the higher-resolution Octomap.

Octomaps of the Azure Kinect box were created while using different resolutions. During the modelling process, the robot and the Azure Kinect box were not moved. To have a first insight, the total occupied volume of the model was calculated. The results are depicted in Table 4.3. It is clearly visible that the occupied volume increases with a lower resolution. This seems to be a good first indicator, that the occupied space in the lower-resolution Octomap overlaps the higher-resolution Octomap. The higher the resolution of the Octomap, the more accurate the occupied space is modeled. The occupied space is then modeled to a minimum. With a resolution of 0.005 m, the occupied volume is 0.00142 m^3 and with a resolution of 0.07 m, the occupied volume is 0.02538 m^3. A first assumption can be made, that the occupied space in the lower-resolution Octomap overlaps the higher-resolution Octomap.

To validate this assumption, the Octomaps of the Azure Kinect box were compared more in depth. Higher-resolution Octomaps were used as references, to see if a previously occupied space has also been modeled as occupied in the lower-resolution Octomap. To calculate the overlapping space, the number of overlapped nodes were divided by the


(a) Octomap with a resolution of 0.1m


(b) Octomap with a resolution of 0.3m


(c) Octomap with a resolution of 0.5m

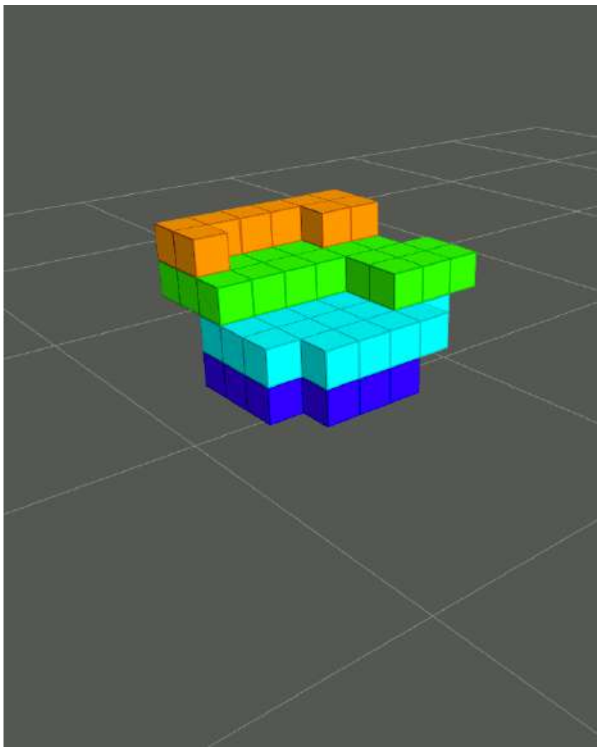
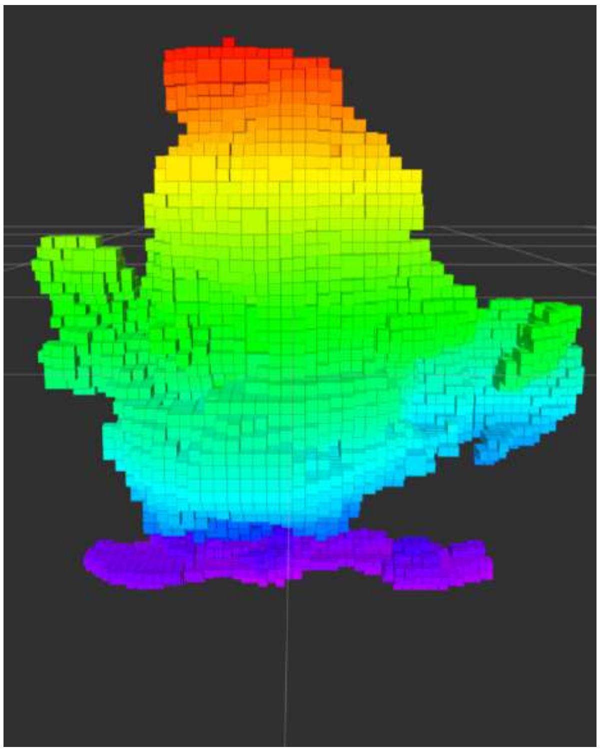
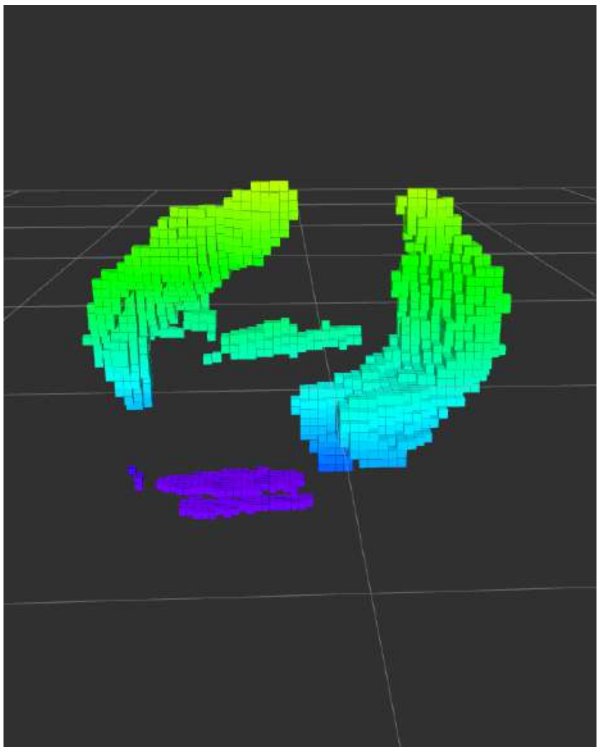

(d) Octomap with a resolution of 0.7m

Figure 4.9: Octomap modelling of the azure kinect camera box with different resolutions


(a) Static Octomap with a resolution of 0.03 m


(b) Dynamic Octomap with a resolution of 0.03 m

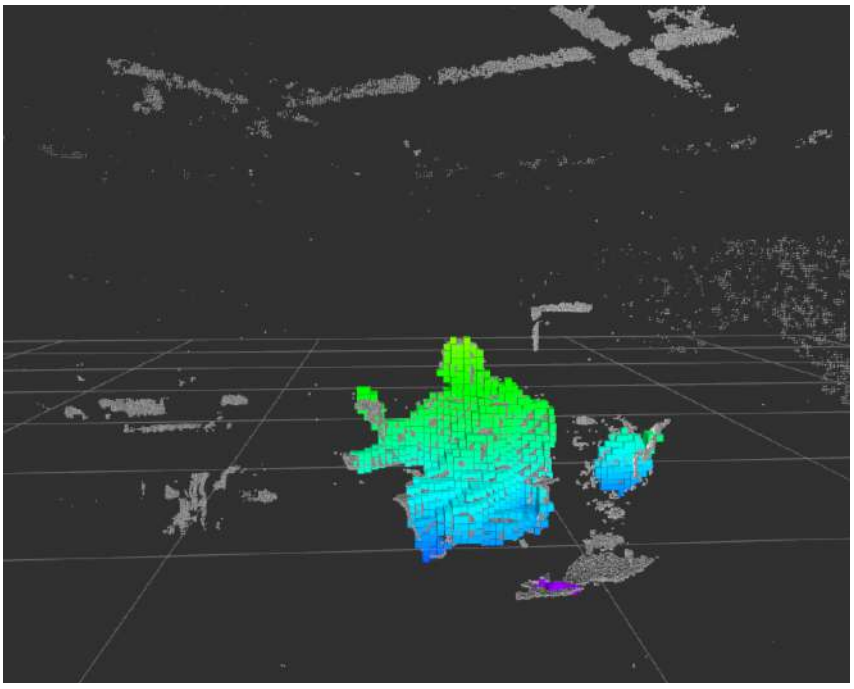

(c) Dynamic Octomap with a resolution of 0.03 m overlapped by the Pointcloud

Figure 4.10: Static and dynamic Octomap while using a rosbag recording

Resolution in m	Total Occupied Volume in m^3
0,005	0.00142
0,01	0.00266
0,03	0.00977
0,05	0.01563
0,07	0.02538

Table 4.3: Total Occupied Volume of the azure kinect camera box with different resolutions

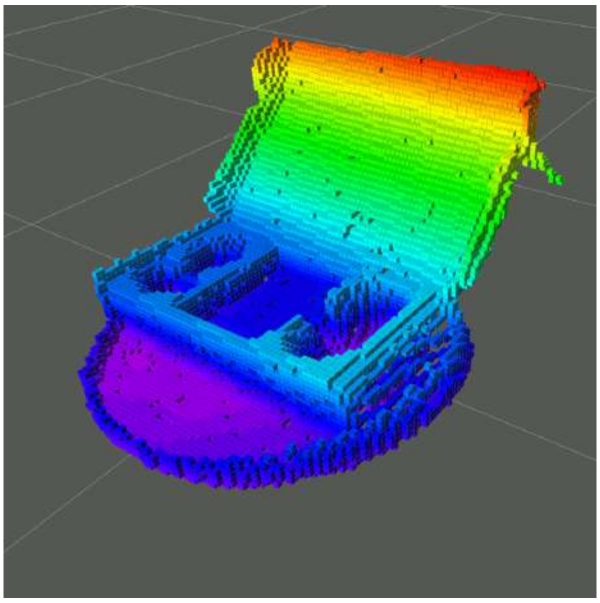
total number of nodes in the higher-resolution Octomap. Overlapped nodes or voxels, are nodes in the higher-resolution Octomap , that are found inside the occupied space of the lower-resolution Octomap. The coordinates of the higher resolution Octomap are compared with the occupied space of the lower resolution Octomap. The Octomaps of the Azure Kinect box that were taken as references can be seen in Figure 4.11. In Figure 4.11a the Octomap of a resolution at 0.005 m and in Figure 4.11b the Octomap of a resolution at 0.01 m is displayed. The results are depicted in table 4.4 and in table 4.5.

Resolution in m	Overlapped Space as percentage	Overlapping Voxels
0,01	94.45 %	13970
0,03	99.38 %	14699
0,05	99.92 %	14779
0,07	100 %	14791

Table 4.4: Comparing Octomap of lower resolution with an Octomap of a resolution of 0.005 m

In table 4.4 the Octomap of a resolution of 0.005 m is used as a reference. The occupied space in the Octomap of a resolution of 0.005 m is modeled with 14791 nodes. All these nodes are found inside the occupied space of the Octomap. The occupied nodes can be found at a different depth level in the octree. The compression technique of the octree cuts off all child nodes of an occupied node. From 14791 Nodes, 13970 Nodes were found inside the occupied space of the Octomap when comparing it with an Octomap of a resolution of 0.01 m. With 94.45 % it is the lowest percentage of overlapping space. This smaller percentage of overlapping space may be linked to greater noise when working with high resolutions. The highest percentage of overlapping space is 100 %, while using an Octomap of a resolution at 0.07 m. When comparing the results in table 4.4, the overlapping space seems to increase with a lower resolution.

In table 4.5 the Octomap of a resolution of 0.01 m is used as a reference. Octomaps with a resolution of 0.03 m, 0.05 m and 0.07 m were compared with this reference Octomap. The Octomap created with a resolution of 0.01 m consist of 3418 occupied nodes. With 3368 nodes, the Octomap of a resolution of 0.03 m has the lowest number of overlapping nodes at 98.54 %. Here again, the percentage of overlapping space seems to increase with a lower resolution. The highest percentage of overlapping space is 99.97 %, while using an Octomap of a resolution at 0.05 m and 0.07 m.


(a) Reference Octomap with a resolution of 0.005 m

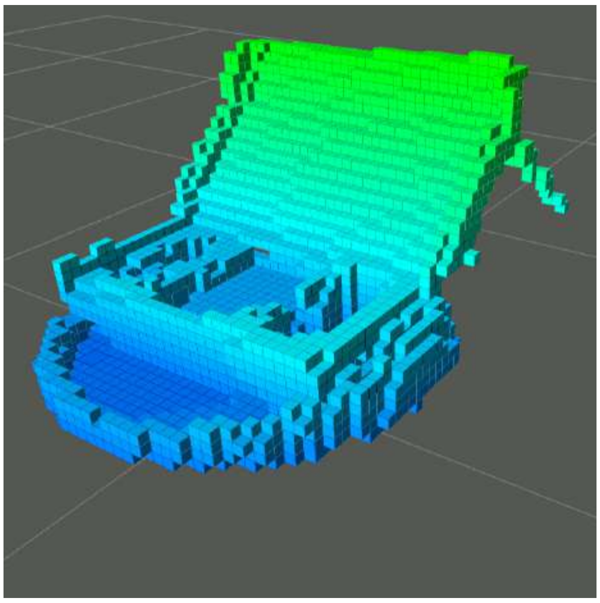

(b) Reference Octomap with a resolution of 0.01 m

Figure 4.11: Reference Octomaps of the azure kinect camera box

Resolution in m	Overlapped Space as percentage	Overlapping Voxels
0,03	98.54 %	3368
0,05	99.97 %	3417
0,07	99.97 %	3417

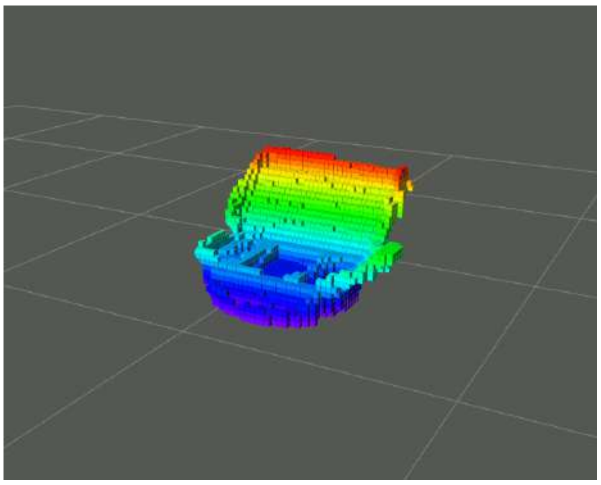
Table 4.5: Comparing Octomaps of lower resolution with an Octomap of a resolution of 0.01 m

Following the evaluation made in table 4.4 and table 4.5, it can be assumed that the occupied space in the lower-resolution Octomap overlaps the higher-resolution Octomap with a high percentage of at least 94 %. When comparing high resolution Octomaps (with 0.005 m and 0.01 m) the overlapping space is minimally lower. This could be linked to greater noise when working with high resolutions. As mentioned before, working with lower resolutions has a positive impact on the reaction time of the robot. For obstacle avoidance and navigation, a resolution of 0.07 m seems to be sufficient. The Octomaps are reliably modeled and the occupied space overlaps the higher-resolution Octomap with a high percentage.

Object placed on	Total Occupied Volume	Total Occupied Node
Stool	0.002761 m^3	2642
Table	0.006862 m^3	6827

Table 4.6: Occupied volume when placing the Azure Kinect box on a stool and a table

Throughout the project, the Azure Kinect camera box served as the object of interest. It was positioned on both a stool and a table. To assess the impact of the base surface on Octomap creation, calculations for both scenarios were made. The occupied volume and the number of occupied nodes for both scenarios are detailed in Table 4.6. In Figure 4.12, visual representations of the Octomap with the camera box on the stool and on the table are provided. Notably, the base surface of the octree varies between these scenarios. As indicated in Table 4.6, when the camera box is positioned on the table, there is an increase in both the occupied volume and the number of occupied nodes. This is primarily due to the larger base surface area offered by the table. To enhance computational efficiency, the base surface should be kept as small as possible. The base surface also affects the working space of the robot. Small base surfaces give more flexibility to the robot to move around an object.


(a) Octomap with a resolution of 0.01 m of the Azure Kinect box placed on a stool

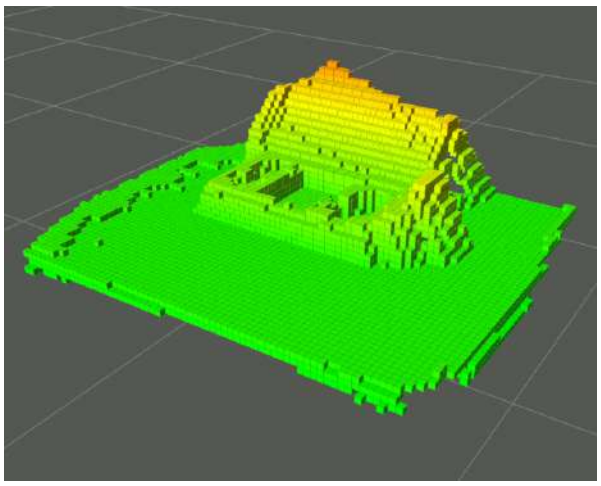

(b) Octomap with a resolution of 0.01 m of the Azure Kinect box placed on a table

Figure 4.12: Octomap of the azure kinect camera box placed on a stool and a table

4.5 Outlook

To sum up, the overall system showcases a good environment modelling set-up to model the object accurately and precisely. Challenges like joint limitations, slow update of Octomaps and calibration issues could be improved and refined. The implementation of NBV with the robot maybe integrated after introducing a validation algorithm which validates the feasible viewpoint poses from the array of best poses. Further, the camera could be calibrated with help of Optitrack Mocap system to extract the position and orientation of camera and reference it to the frame of end effector on the robot. Moreover, integrating some techniques for adaptive learning and real time adjustments of parameters could also increase the overall performance. This experiment provides a good base for future research and implementations in the field of environment modelling around robotics manipulators by integrating Next best view algorithm and object detection algorithm.

Disclaimer:

For the purpose of enhancing linguistic clarity, coherence, grammar and overall structure, parts of this chapter have been reviewed and refined using ChatGPT, an artificial intelligence language model developed by OpenAI. While ChatGPT can generate and improve human-like text, it doesn't inherently validate the accuracy or factual content of the report. We used ChatGPT to help us develop ROS nodes and structure the latex document.In 4.4 ChatGPT was used to enhance the linguistic clarity of the section.

5 Conclusion

In conclusion, the proper modeling of the environment is essential for the successful operation of autonomous systems, such as automated driving and industrial robots. These systems rely on accurate and comprehensive information about their surroundings to make informed decisions and interact effectively with their environments.

Throughout this report, we have explored various methods and techniques for modeling the environment in the context of automated driving and robotics. From recreating scenarios in Unreal Engine 5.1 to three-dimensional mapping and occupancy modeling using the UR10 robot and Microsoft Azure Kinect camera, each chapter has provided valuable insights into the challenges and opportunities in environment modeling.

One key takeaway from this experimentation is the significance of object recognition and detection. Object recognition plays a crucial role in understanding and interacting with the environment. Whether it's identifying pedestrians and vehicles in adverse weather conditions or mapping objects in three dimensions, accurate object modeling is fundamental. It enables collision avoidance, path planning, and overall system reliability.

In conclusion, as we continue to advance autonomous systems, the importance of proper object recognition and detection cannot be overstated. It is a critical component in ensuring the optimal performance and safety of these systems in the complex and dynamic world they operate in. Therefore, further research and development in this area are essential for the continued improvement of autonomous technologies.

Bibliography

Aleotti, J., D. Lodi Rizzini, R. Monica, and S. Caselli (Sept. 2014): "Global Registration of Mid-Range 3D Observations and Short Range Next Best Views". In: *IEEE International Conference on Intelligent Robots and Systems.*

Bircher, A., M. S. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart (May 2016): "Receding Horizon "Next-Best-View" Planner for 3D Exploration". In: pp. 1462–1468.

Bochkovskiy, A., C. Wang, and H. M. Liao (2020): "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: *CoRR* abs/2004.10934. arXiv: `2004.10934`. URL: `https://arxiv.org/abs/2004.10934`.

Deng, J., S. Shi, P. Li, W. Zhou, Y. Zhang, and H. Li (2021): "Voxel R-CNN: Towards High Performance Voxel-based 3D Object Detection". In.

Ding, Y., Z. Li, and D. Yastremsky (May 2021): "Real-time Face Mask Detection in Video Data". In.

Engine 5.1 Documentation (n.d.). URL: `https://docs.unrealengine.com/5.1/en-US/`.

Engine Materials Tutorials | Unreal Engine 5.1 Documentation (n.d.). URL: `https://docs.unrealengine.com/5.1/en-US/unreal-engine-materials-tutorials/`.

Everingham, M., L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman (June 1, 2010): "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88.2, pp. 303–338. URL: `https://doi.org/10.1007/s11263-009-0275-4` (visited on 09/27/2023).

Geiger, A., P. Lenz, C. Stiller, and R. Urtasun (2013): "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR).*

Hornung, A., K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard (Apr. 1, 2013): "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees". In: *Autonomous Robots* 3. Software available at `http://octomap.github.com`, pp. 189–206.

Intersection over Union (IoU) for object detection (2023). URL: `https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/` (visited on 09/27/2023).

Kim, Y., K. Park, M. Kim, D. Kum, and J. W. Choi (2023): "3D Dual-Fusion: Dual-Domain Dual-Query Camera-LiDAR Fusion for 3D Object Detection". In.

Kübel, J. and J. Brandes (2022): “Camera-Radar Sensor Fusion using Deep Learning”. Available: https://hdl.handle.net/20.500.12380/305503. MA thesis. Chalmers University of Technology.

Lin, T.-Y., M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár (2014): *Microsoft COCO: Common Objects in Context.* URL: `http://arxiv.org/abs/1405.0312`.

Mcgreavy, C., L. Kunze, and N. Hawes (Dec. 2016): “Next Best View Planning for Object Recognition in Mobile Robotics”. In.

Nabati, R. and H. Qi (2020): “CenterFusion: Center-based Radar and Camera Fusion for 3D Object Detection”. In: *CoRR* abs/2011.04841. arXiv: `2011.04841`. URL: `https://arxiv.org/abs/2011.04841`.

OpenPCDet (2023). URL: `https://github.com/open-mmlab/OpenPCDet` (visited on 09/27/2023).

Palffy, A., E. Pool, S. Baratam, J. F. P. Kooij, and D. M. Gavrila (2022): “Multi-Class Road User Detection With 3+1D Radar in the View-of-Delft Dataset”. In: *IEEE Robotics and Automation Letters* 7.2, pp. 4961–4968.

perception_pcl (2023). URL: `https://github.com/ros-perception/perception_pcl` (visited on 09/02/2023).

Salton, G. and M. J. McGill (1986): “Introduction to Modern Information Retrieval”, McGraw-Hill, Inc., New York, NY, USA,” in: *International Journal of Computer Vision.*

Simonelli, A., S. R. R. Bulò, L. Porzi, M. López-Antequera, and P. Kontschieder (May 29, 2019): “Disentangling Monocular 3D Object Detection”. In: arXiv:1905.12365. arXiv: `1905.12365[cs]`. URL: `http://arxiv.org/abs/1905.12365` (visited on 09/27/2023).

Triebel, R., P. Pfaff, and W. Burgard (Oct. 2006): “Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing”. In: pp. 2276–2282.

Universal Robots (2023). URL: `https://www.universal-robots.com/` (visited on 09/01/2023).

Wilhelms, J. and A. Van Gelder (July 1, 1992): “Octrees for Faster Isosurface Generation”. In: *ACM Trans. Graph.* 11.3, pp. 201–227. URL: `https://doi.org/10.1145/130881.130882`.

Yu, F., D. Wang, E. Shelhamer, and T. Darrell (2018): “Deep Layer Aggregation”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.* IEEE, pp. 2403–2412.

Zhang, W., Z. Wang, and C. C. Loy (2021): “Exploring Data Augmentation for Multi-Modality 3D Object Detection”. In.

Zhou, X., V. Koltun, and P. Krähenbühl (2020): “Tracking Objects as Points”. In: *ECCV.*

Zhou, X., D. Wang, and P. Krähenbühl (2019a): “Objects as Points”. In: *arXiv preprint arXiv:1904.07850.*

Zhou, X., D. Wang, and P. Krähenbühl (2019b): “Objects as points”. In: *arXiv preprint arXiv:1904.07850.*

6 Appendix

This appendix provides 15 scenes out of the 20 scenes created in Unreal Engine that were not elaborated on within the main part of the report. Please know that the numbering is just for the sake of reference and has no additional meaning.

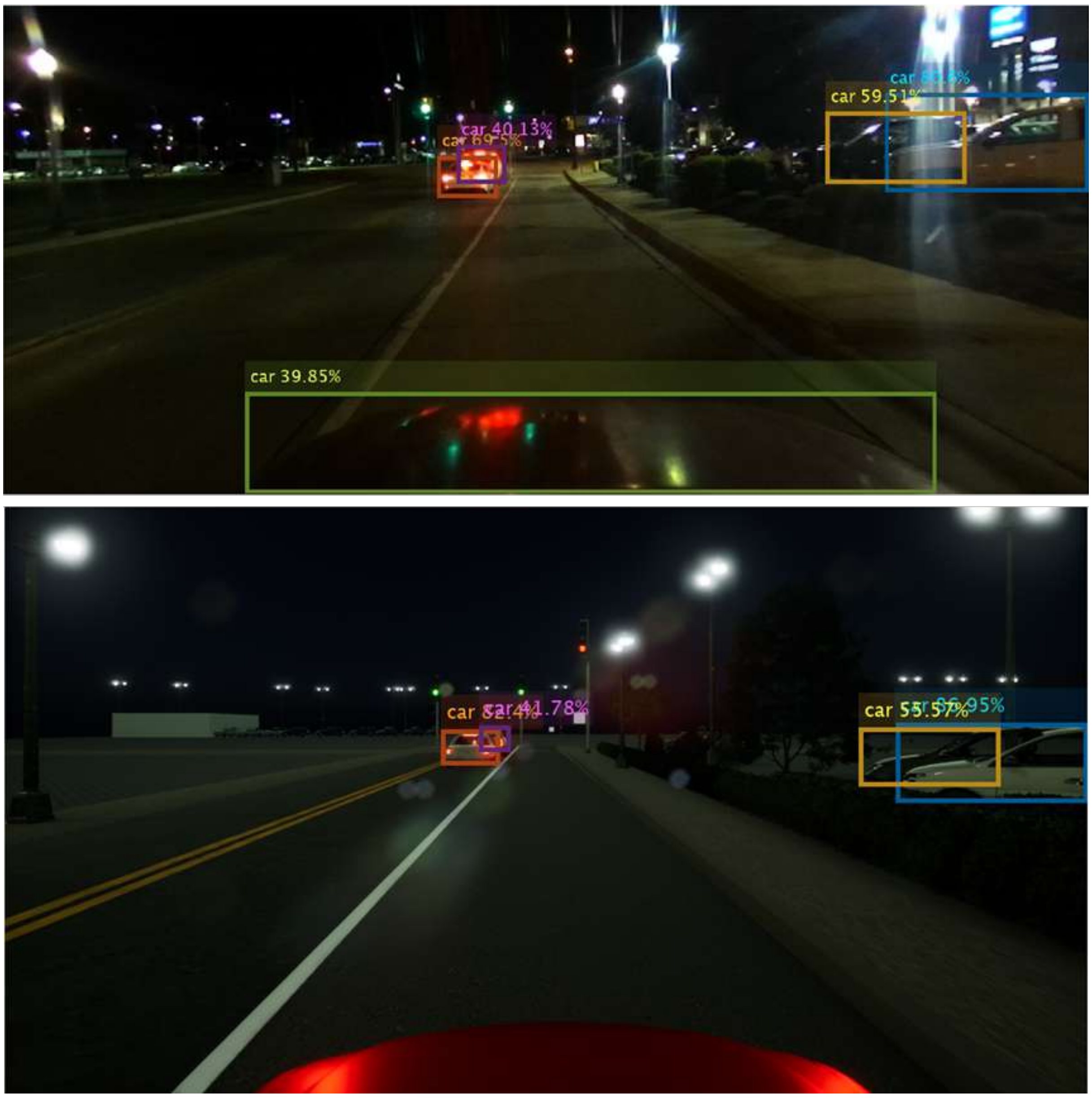


Figure 6.1: Scene 1

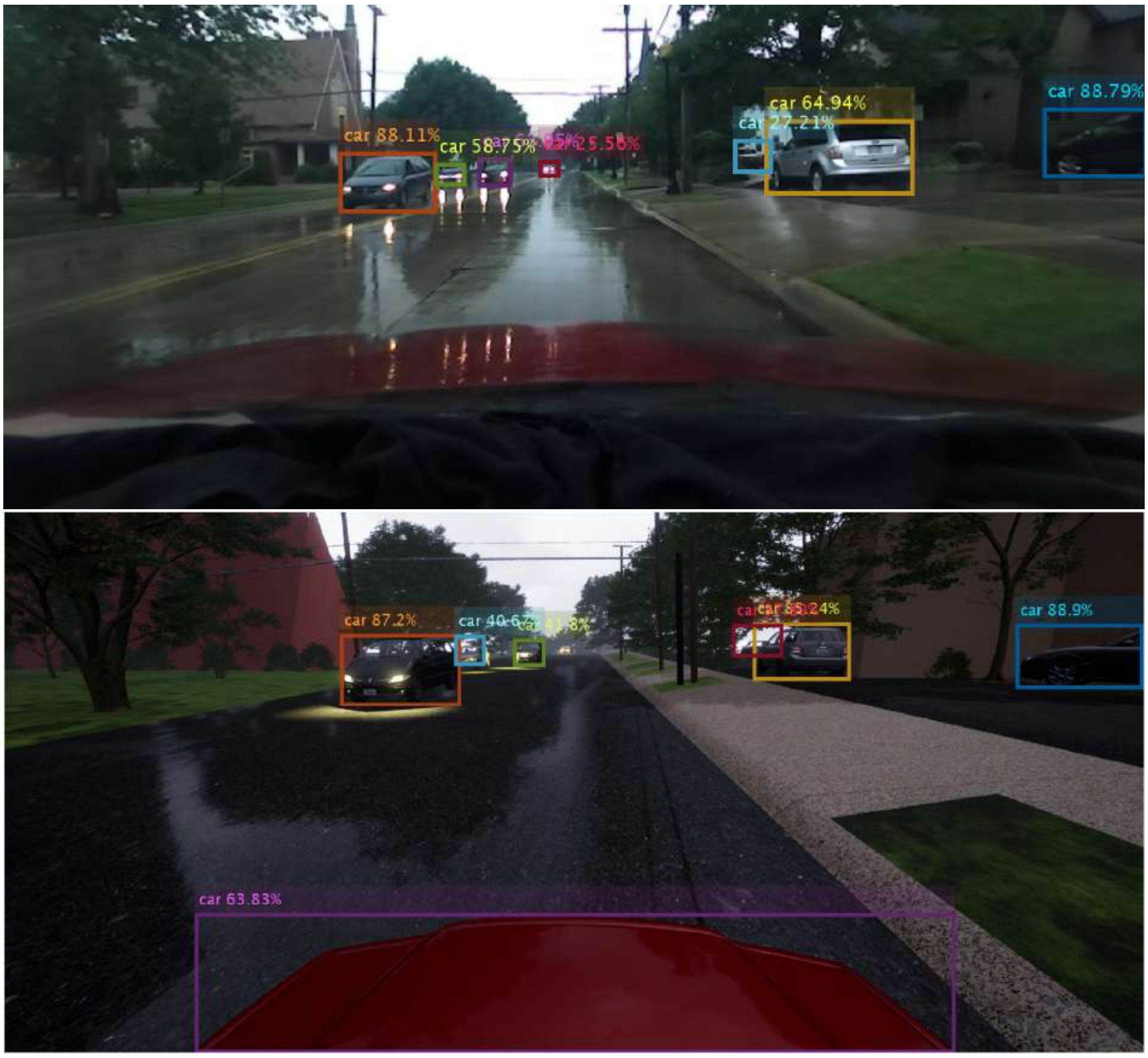


Figure 6.2: Scene 2

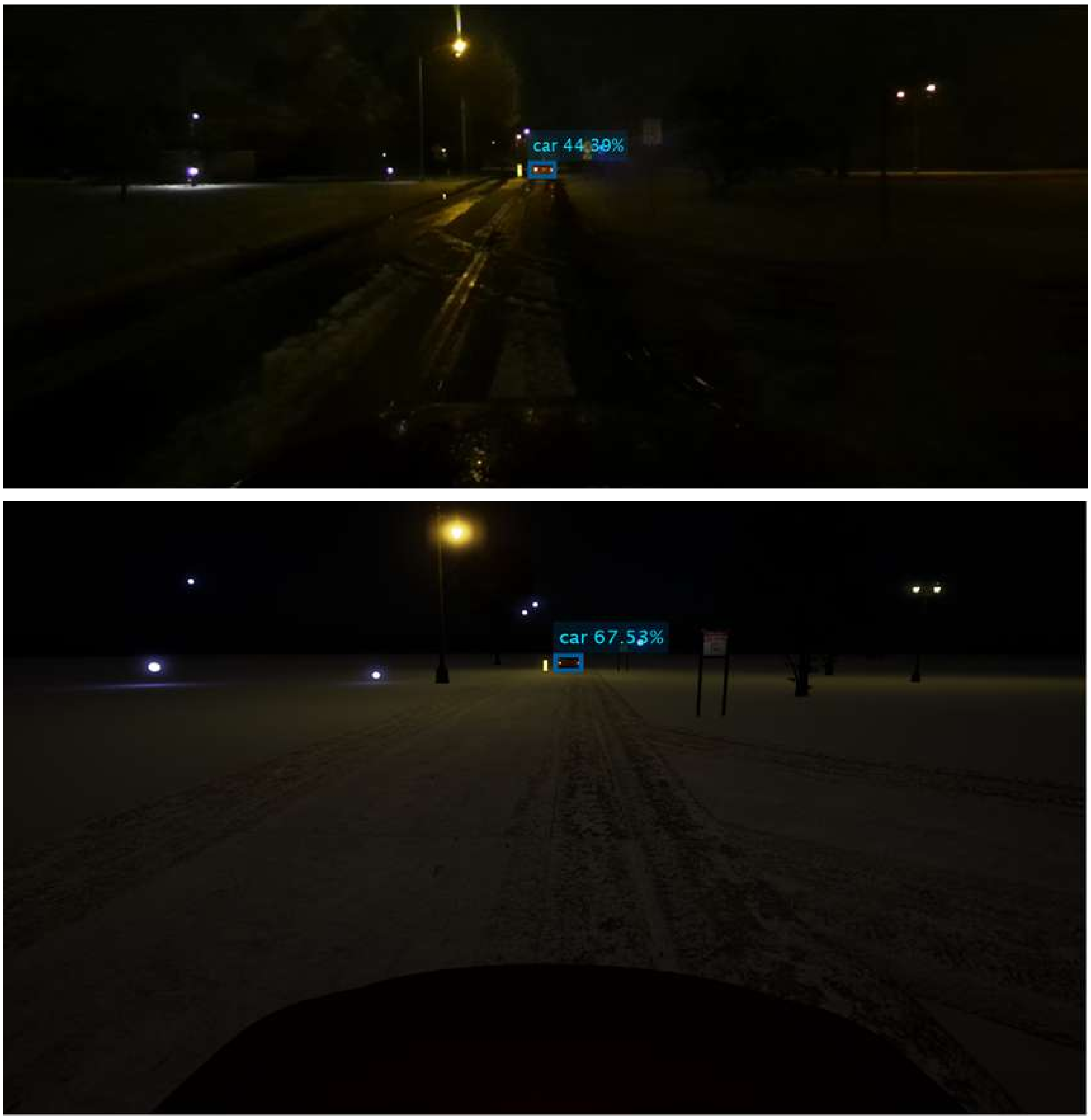


Figure 6.3: Scene 3

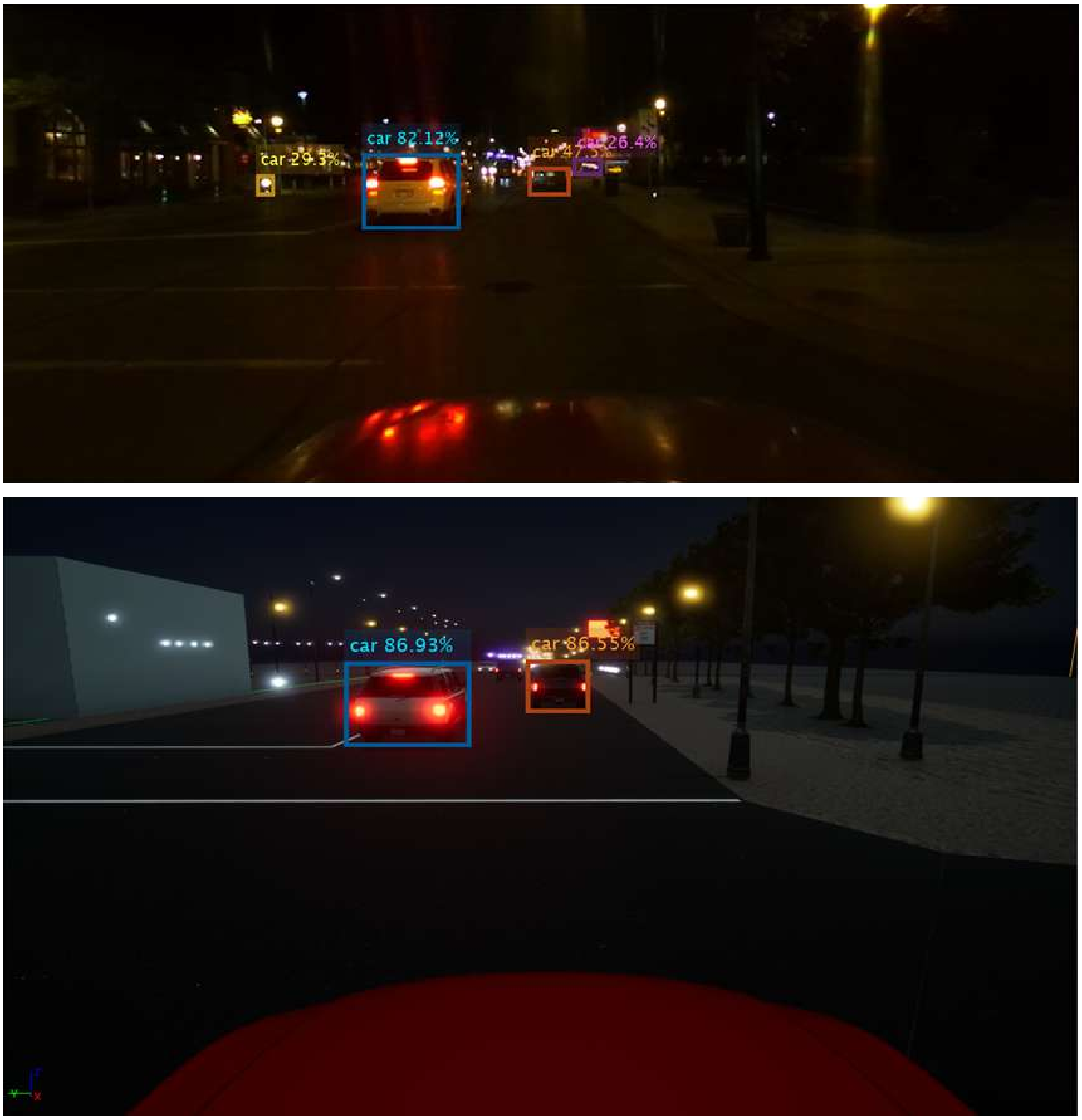


Figure 6.4: Scene 4

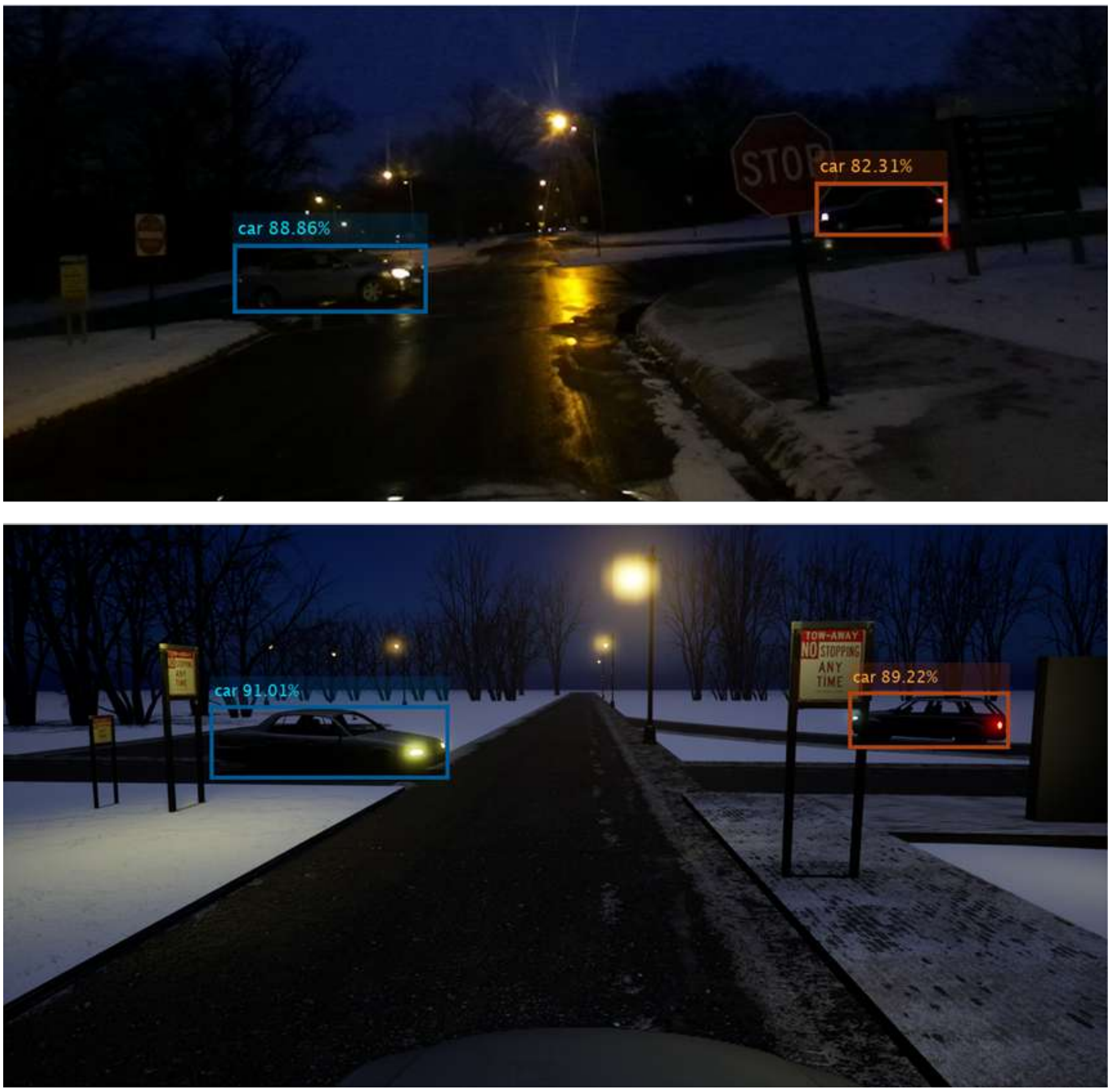


Figure 6.5: Scene 5

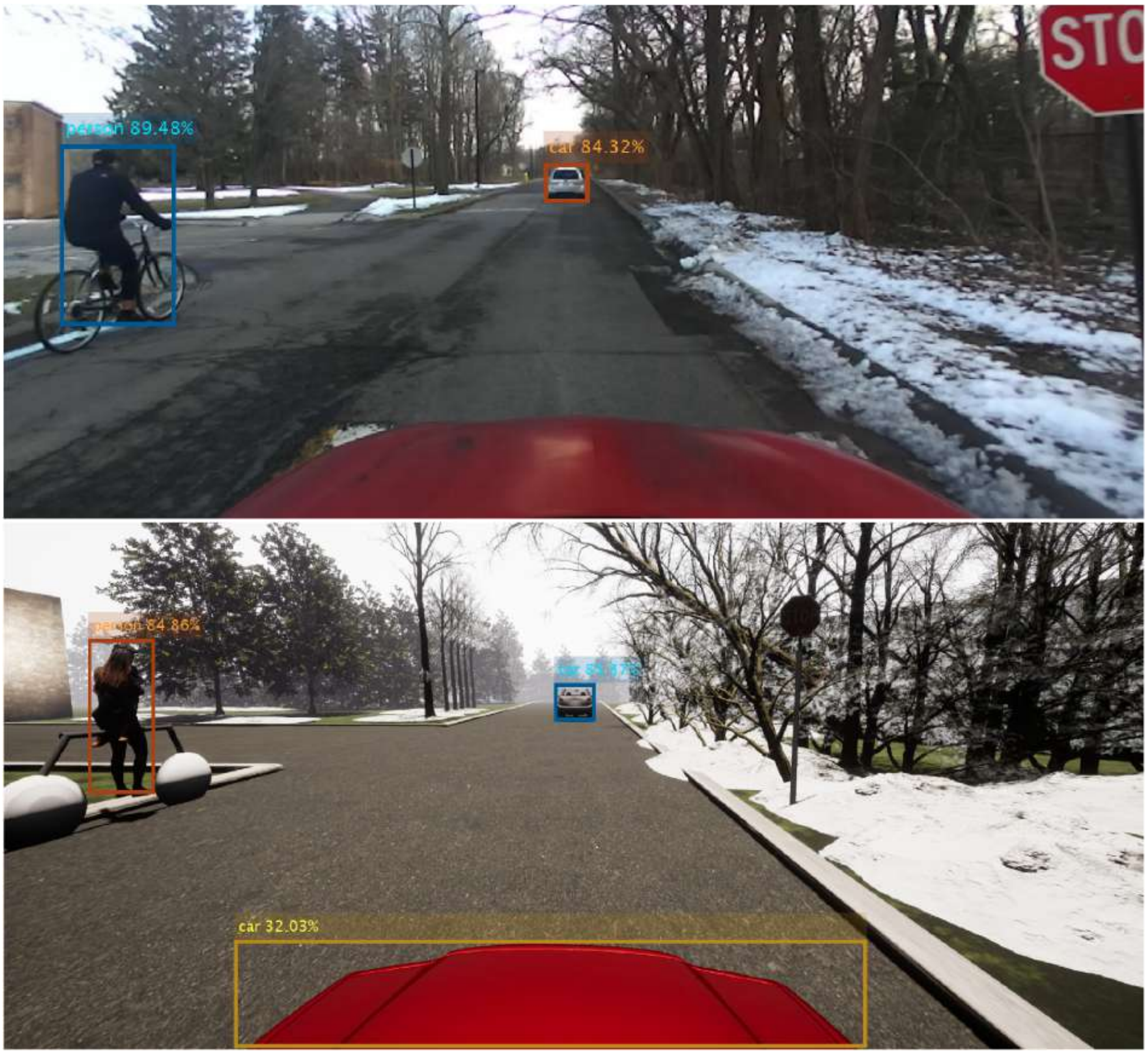


Figure 6.6: Scene 6

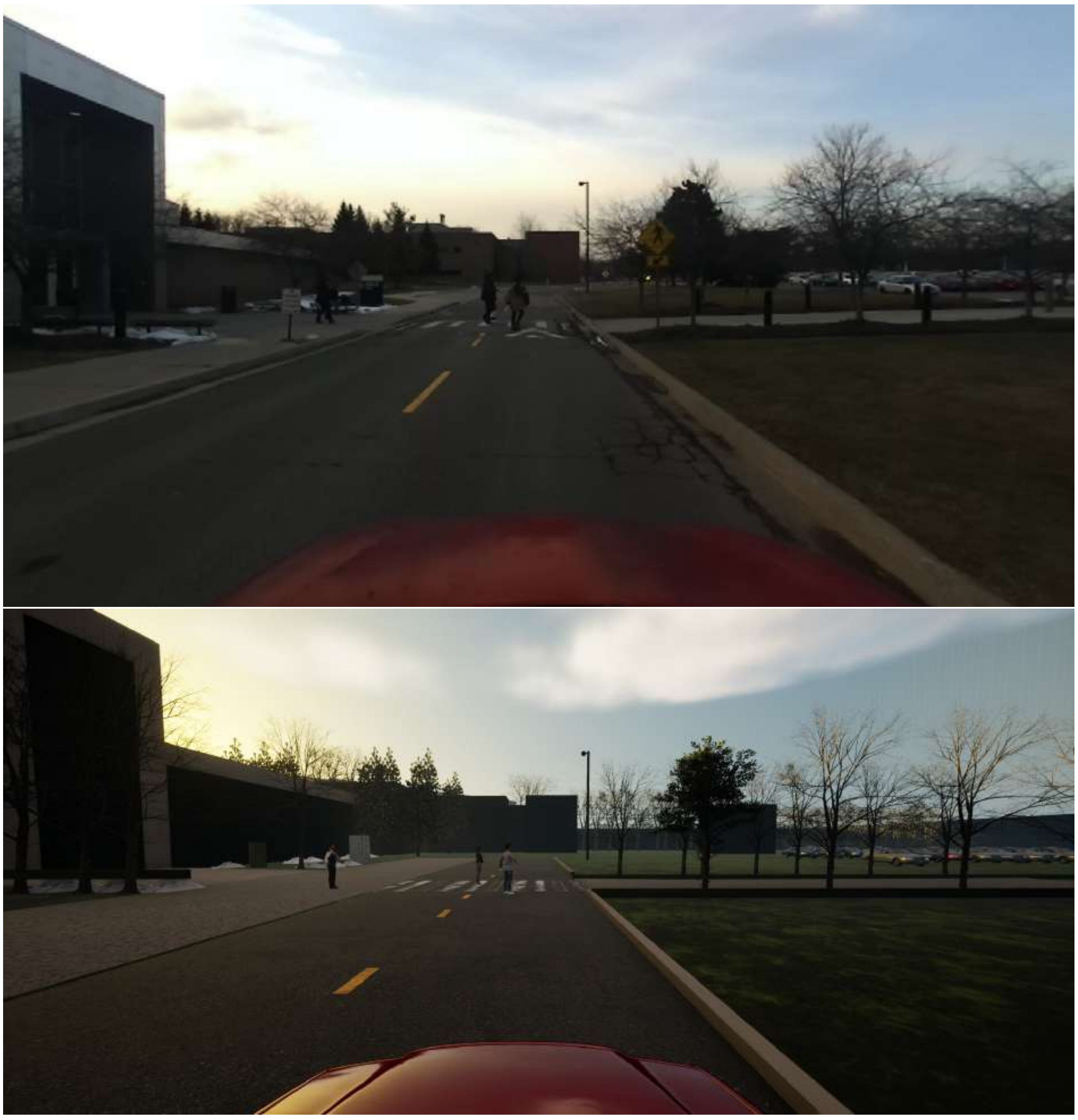


Figure 6.7: Scene 7

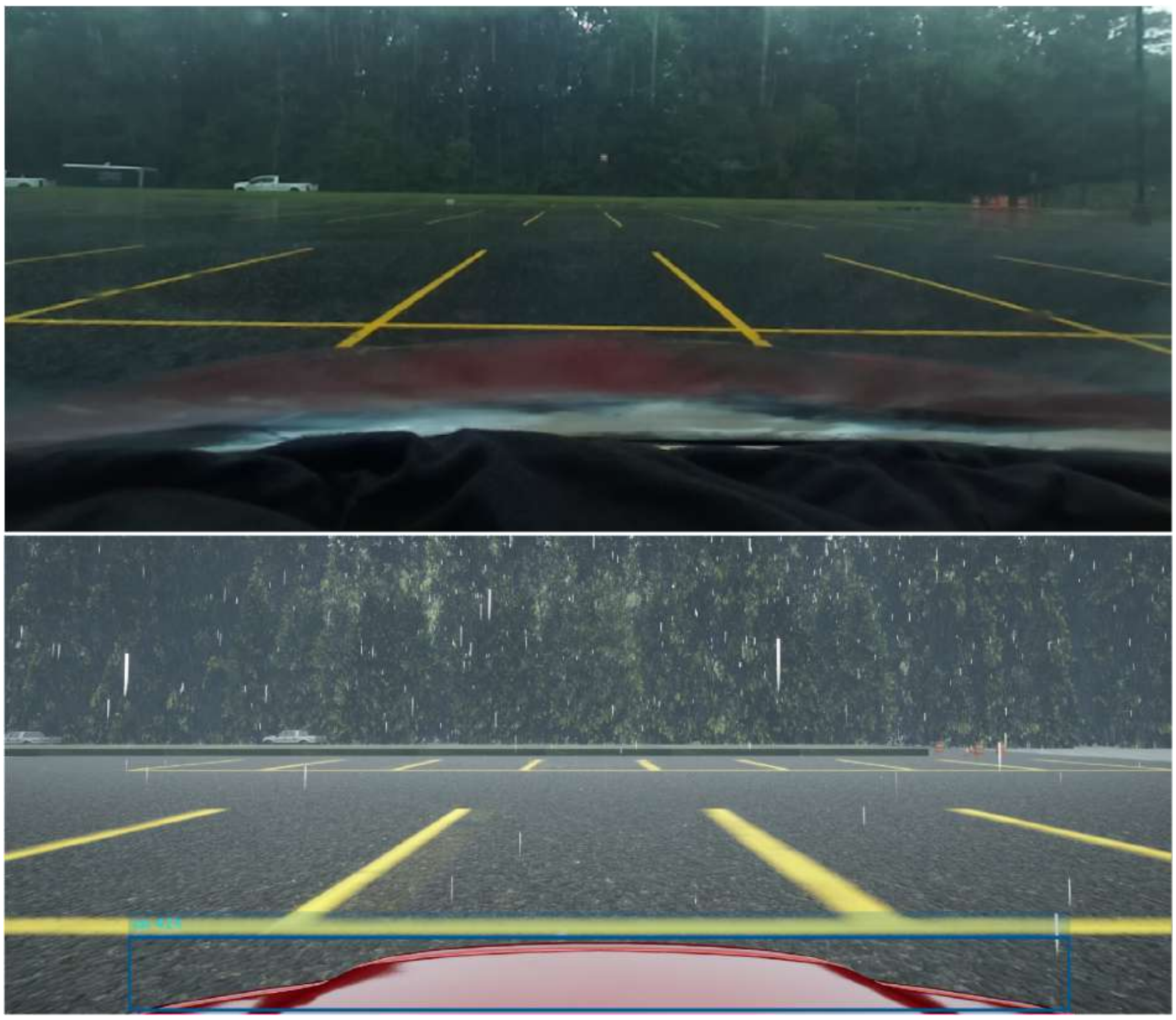


Figure 6.8: Scene 8

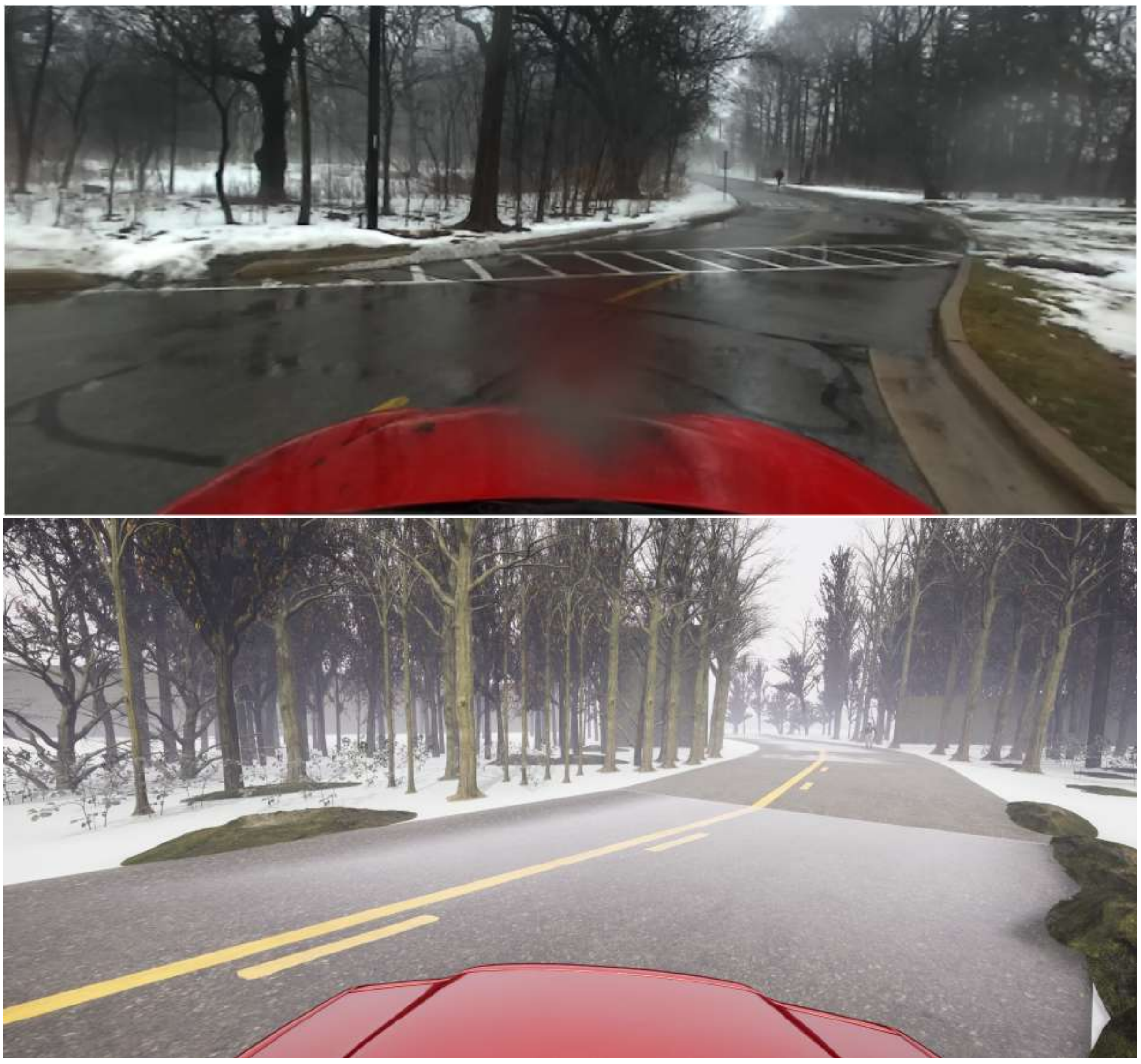


Figure 6.9: Scene 9

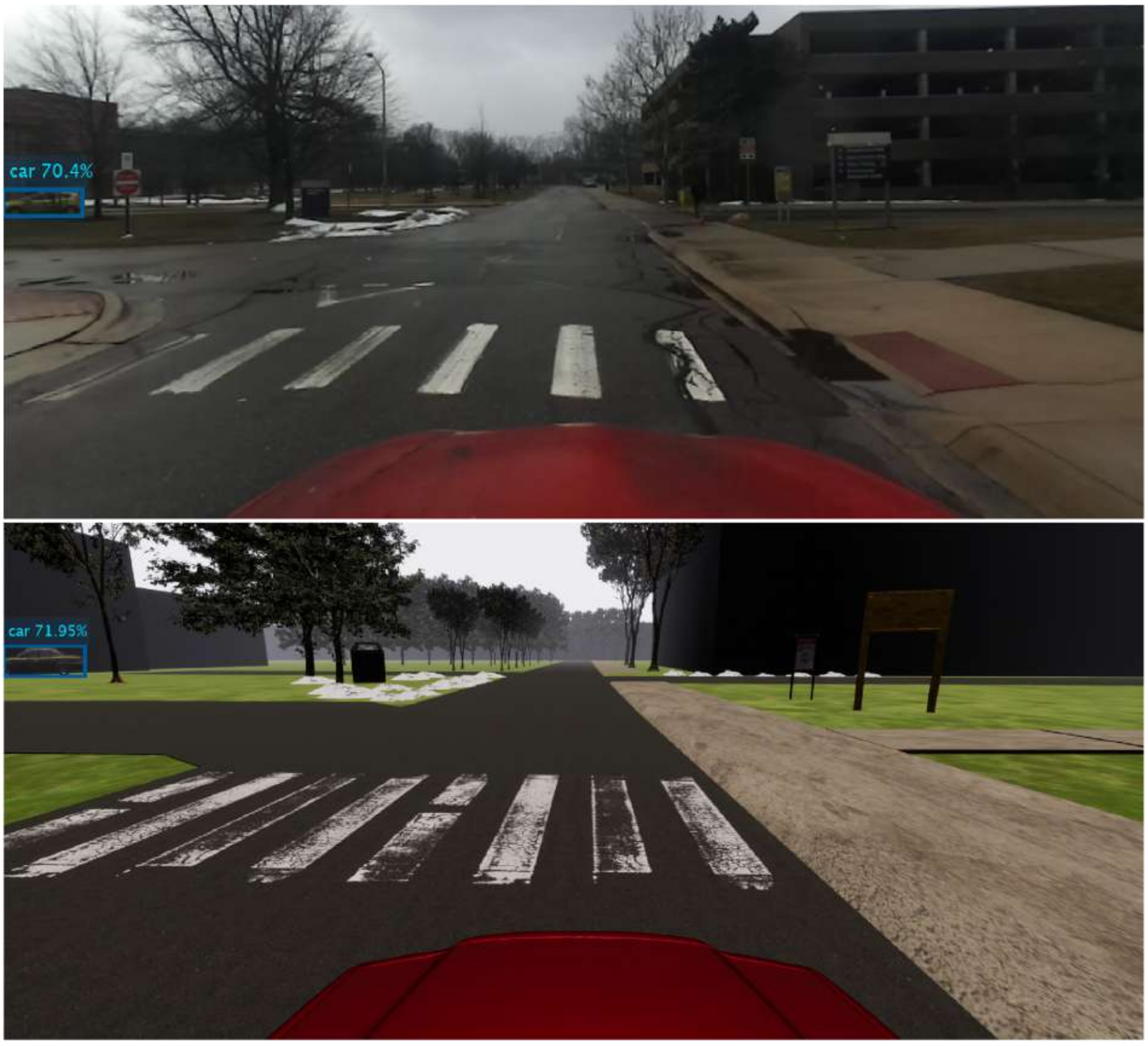


Figure 6.10: Scene 10

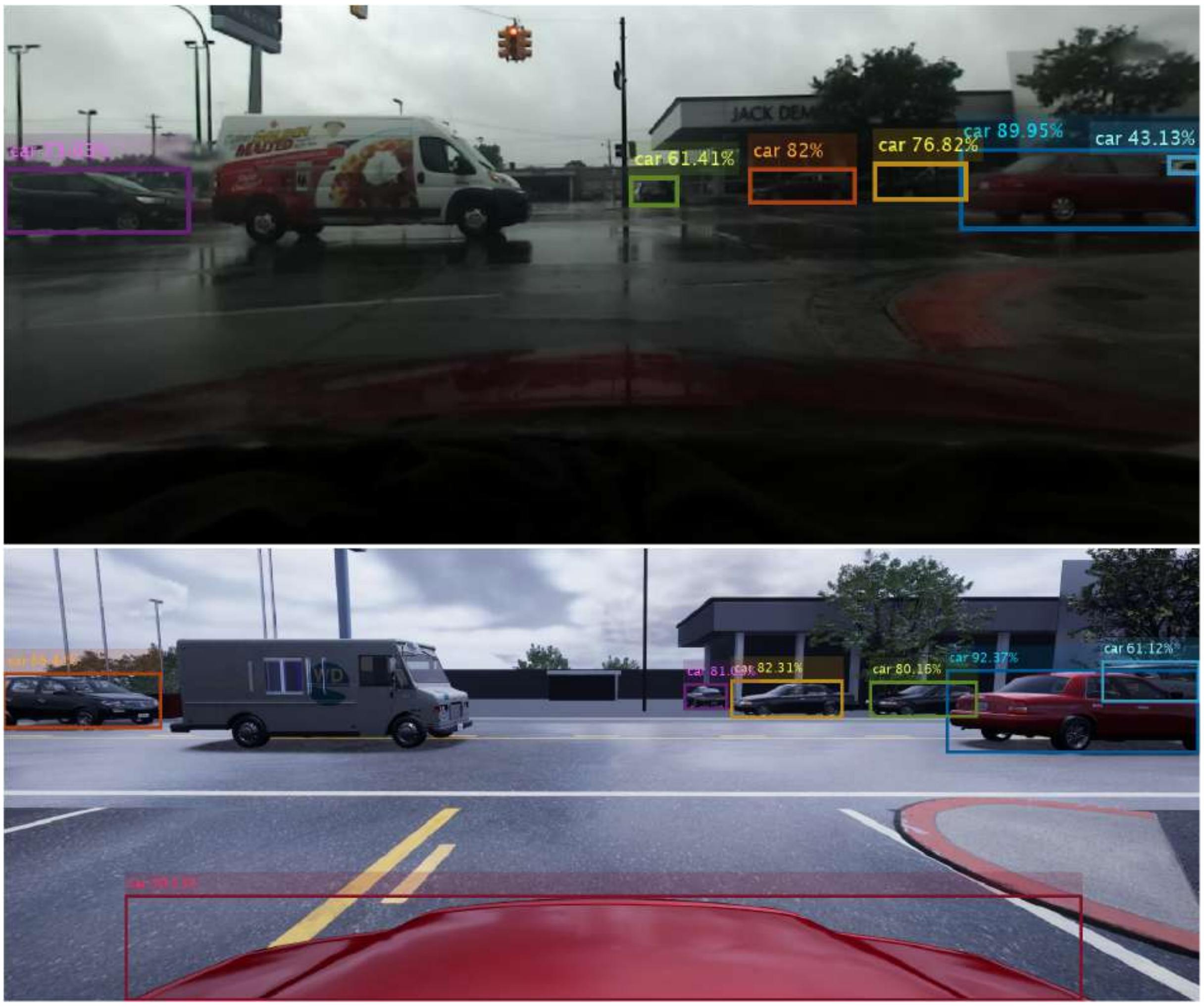


Figure 6.11: Scene 11

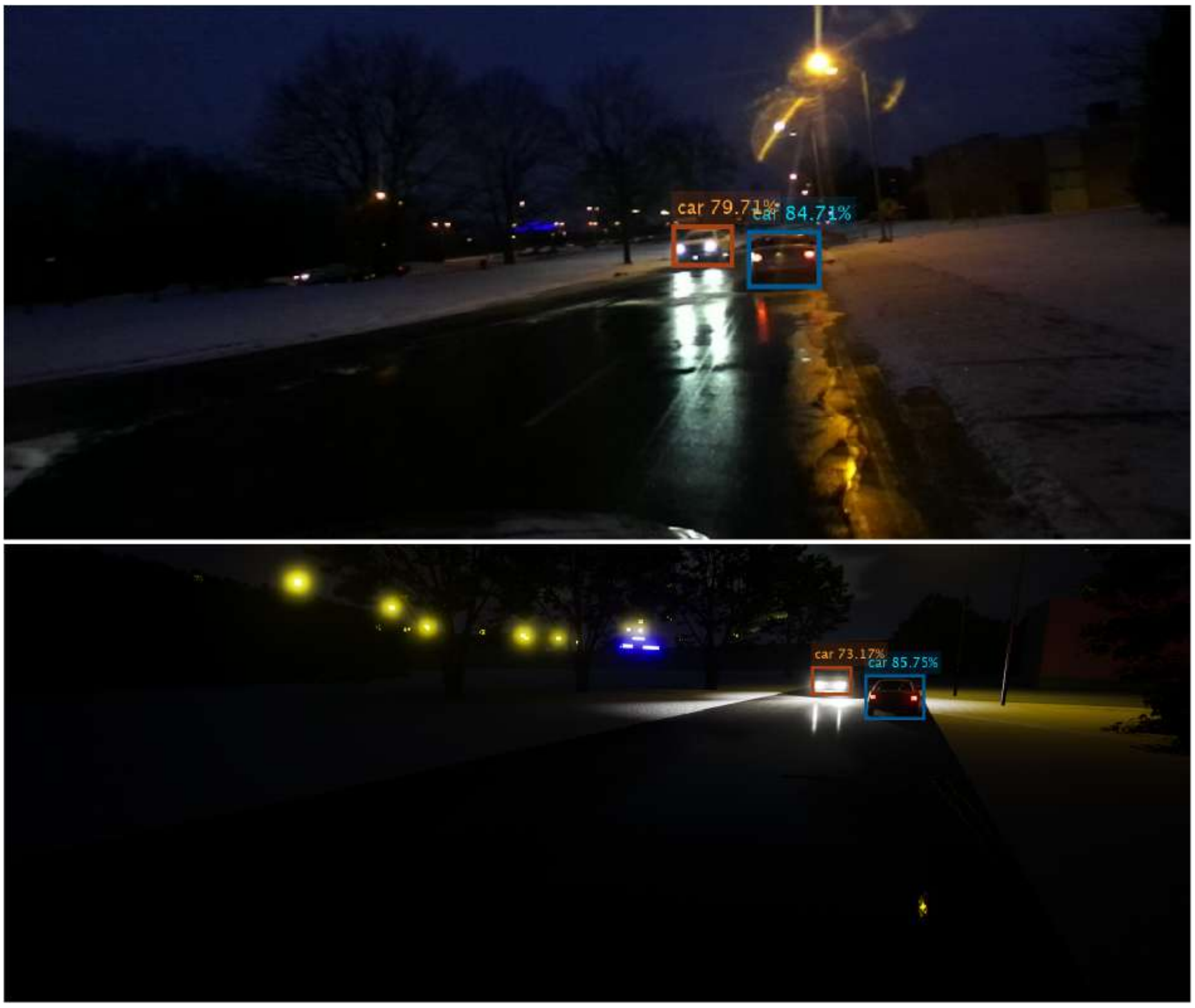


Figure 6.12: Scene 12

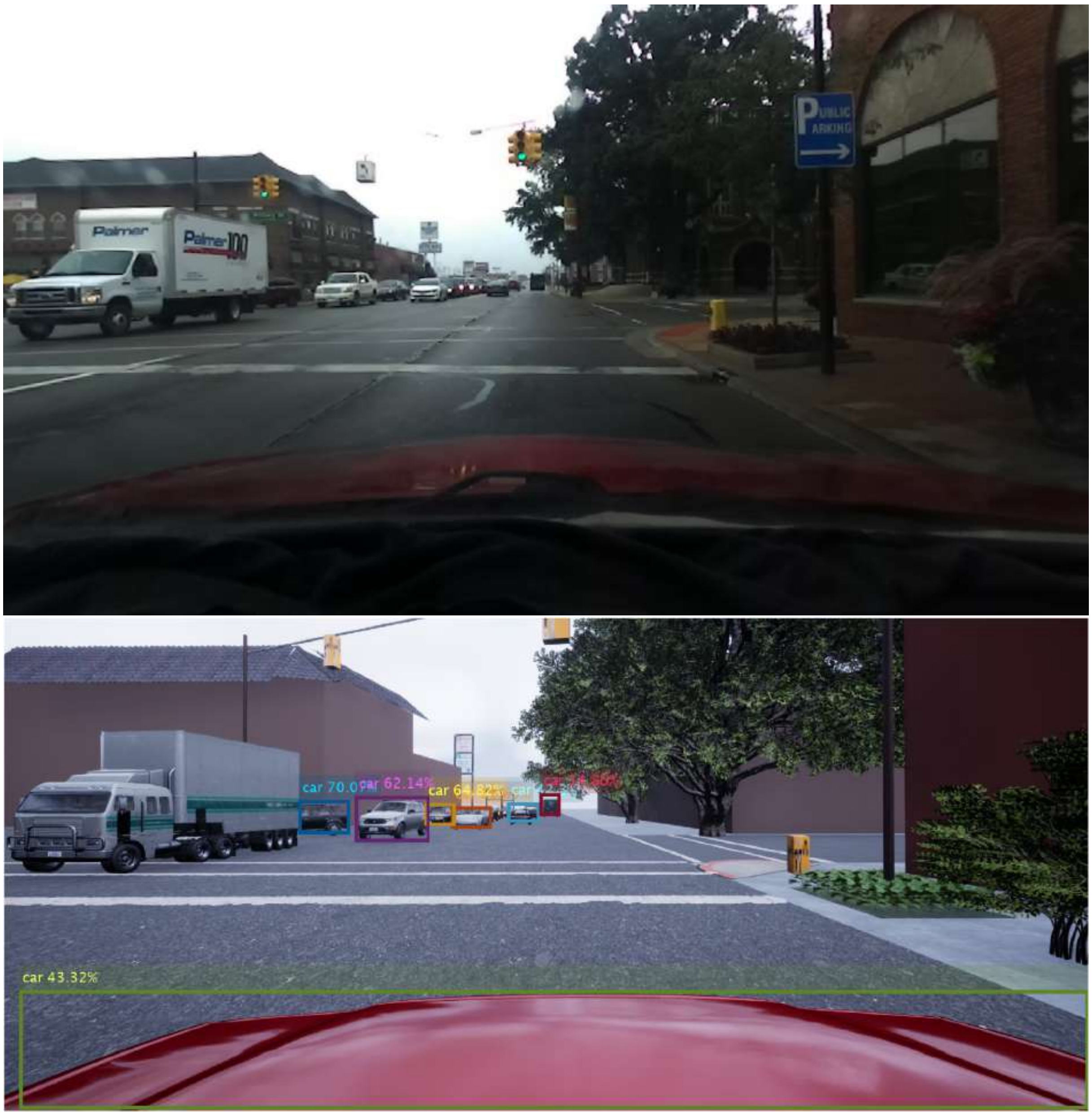


Figure 6.13: Scene 13

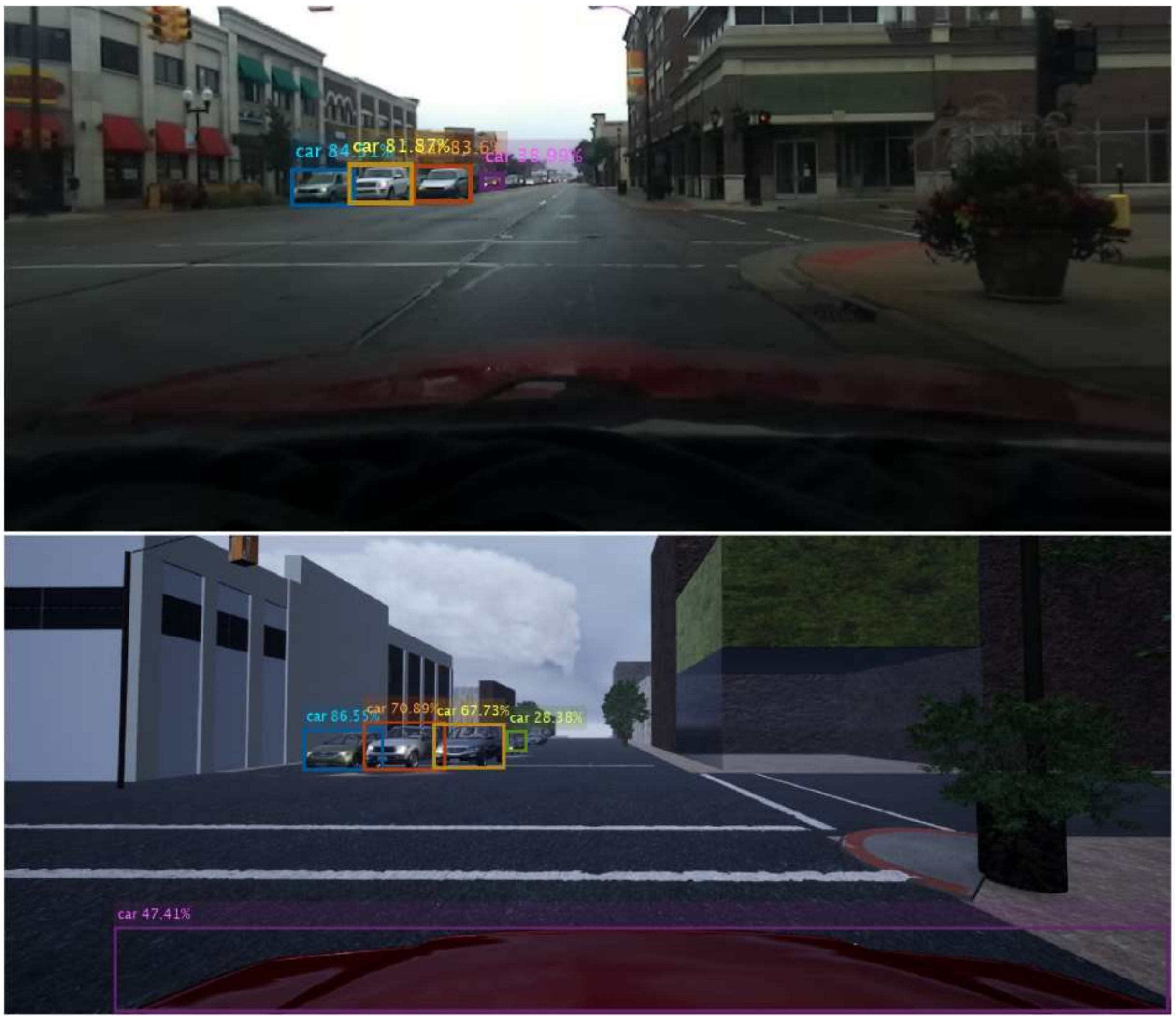


Figure 6.14: Scene 14

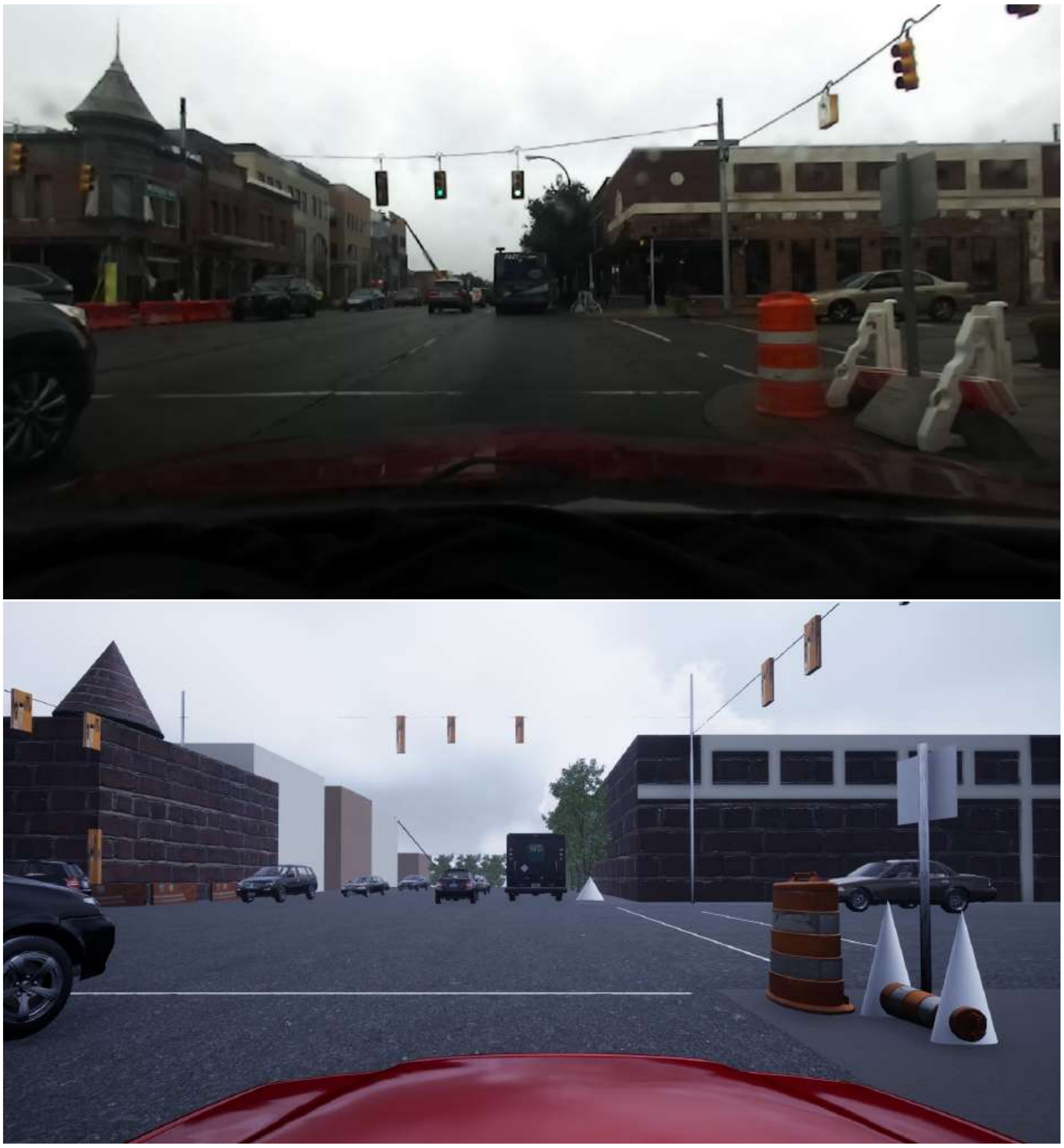


Figure 6.15: Scene 15