Department of Informatics
Technical University of Munich

Bachelor's Thesis in Informatics

Real-Time Monocular 3D Object Detection to Support Autonomous Driving

Echtzeitfähige und Kamera-basierte 3D Objektdetektion für die Unterstützung des Autonomen Fahrens

Supervisor Prof. Dr.-Ing. habil. Alois C. Knoll

Advisor Walter Zimmer, M.Sc.

Author Leon Blumenthal

Date February 15, 2022 in Garching

Disclaimer

I confirm that this Bachelor's Thesis is my own work and I have documented all sources and material used.

Garching, February 15, 2022

(Leon Blumenthal)

Abstract

Autonomous driving is one of the most influential technologies of our time. It will decrease accidents on the road and save countless lives. However, intelligent infrastructure systems are needed to aid self-driving cars by providing crucial information from outside the limited range of vehicle-mounted sensors. The 3D detection of vehicles on the road is one of the use cases. In contrast to costly *LiDAR* sensors which are commonly used for this, cameras can pose a cost-effective alternative, however, they only capture 2D information. Therefore, intelligent algorithms are essential to overcome this issue.

This thesis proposes a real-time approach to estimate 3D bounding boxes for vehicles within the *Providentia++* test stretch, which contains straight highway segments and precisely calibrated cameras. Leveraging these constraints, the method projects the bottom contour of instance masks, generated with existing instance segmentation models, from the image onto the ground plane. These should approximate exactly two bottom edges of the ideal 3D bounding boxes and are therefore used to estimate the location and dimensions of the vehicles. Additionally, the heights are estimated based on a simple geometric argument involving the instance mask the heights and the viewing angle of the camera. The proposed approach is also evaluated on the *A9-Dataset* and produces results that surpass the previously evaluated methods for monocular 3D object detection.

Zusammenfassung

Autonomes Fahren ist eine der einflussreichsten Technologien in der heutigen Zeit. Es wird die Zahl der Autounfälle vermindern und unzählige Leben retten. Jedoch wird hierfür ebenfalls intelligente Verkehrsinfrastruktur benötigt, welche die selbstfahrenden Fahrzeuge mit Daten unterstützen, die von außerhalb der Sensorreichweiten stammen. Einer dieser Anwendungsfälle ist die 3D Objektdetektion von Straßenfahrzeugen. Im Gegensatz zu *LiDAR*-Sensoren sind Kameras eine kostengünstige Alternative, welche aber nur 2D Daten aufnehmen. Um dieses Problem zu lösen, sind intelligente Algorithmen essenziell.

In dieser Arbeit wird eine echtzeitfähige Methode vorgeschlagen, welche 3D Bounding-Boxen von Fahrzeugen innerhalb der *Providentia++*-Teststrecke erkennen kann. Die Teststrecke beinhaltet gerade Autobahnabschnitte mit präzise kalibrierten Kameras. Mithilfe dieser können die unteren Konturen einer Maske, welche von einem Instance-Segementation-Modell erstellt wurde, vom Kamerabild auf die Straßenebene projiziert werden. Diese sollten dann genau zwei untere Kanten der idealen 3D-Bounding-Boxen approximieren und können so genutzt werden, um die Abmessungen und Positionen der Fahrzeuge zu bestimmen. Zusätzlich werden die Fahrzeughöhen mit einem einfachen geometrischen Argument, welches die Maskenhöhen und den Kamerawinkel miteinbezieht, bestimmt. Die Methode wird ebenfalls auf dem *A9-Dataset* evaluiert und liefert bessere Erbenisse als zuvor getestete Ansätze.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Autonomous driving is one of the most exciting technologies that is being developed in our time. It will decrease the number of accidents on the road by removing human error, which in turn will save countless lives. Additionally, traffic will be more efficient due to intercommunication between vehicles and road infrastructure. Also, humans can be more productive since less time is being spent steering a vehicle. Furthermore, multiple users can share vehicles, which reduces the demand for producing new vehicles and decreases the total number of vehicles in a city. Therefore, fewer parking spaces are required, which allows for more natural space. However, technology in this domain is still in its infancy, yet it is a very active field of research. Autonomous vehicles (*AVs*) have multiple sensors onboard (e.g. *LiDAR*, radar, and cameras). These can perceive their local environment and based on that, intelligent decisions can be made. This is generally called the vehicle-side or ego perspective.

However, especially on highways, the limited range of vehicle-mounted sensors, which can only capture information in a small area around them, is not enough. For example, the vehicle directly in front occludes all other vehicles ahead in the same lane. This limitation can be overcome by intelligent infrastructure systems, that aid the vehicles by providing additional information about the current traffic situation and possible hazards. Generally, capturing information from outside a vehicle at a fixed location is called the road-side perspective. With a single infrastructure system, that all vehicles on the same road segment communicate with, it is also possible to connect the different vehicles, such that they can act as a symbiotic system and therefore cause fewer disruptions to the traffic flow, decrease congestion, and improved efficiency in general.

Exactly such an intelligent infrastructure system is studied in the research project *Providentia++* [Kra+]. Multiple sensors of different types were installed at various locations, mainly gantry bridges and masts, on a test stretch of the *A9* highway near Munich. The goal of the project is to detect all vehicles of different types on the test stretch with various sensors and combine them all into a *digital twin* of the traffic on the road. This data can then be sent back to the individual vehicles. As the next part of the project, adding urban areas to the *digital twin* is researched.

Regardless of ego or road-side perspective, the first step is generally to get an accurate representation of the current state of the environment. This is commonly called *Perception*. The accuracy of it is of crucial importance because all errors will be transferred into the next steps, which cannot compromise them. Therefore it is the foundation of any use case in this domain. Generally, vehicles are the most important object to perceive, since they are normally the only moving entities on the road. 3D object detection is the task that estimates their locations and dimensions. Multiple types of sensors can be employed for this, commonly

LiDAR sensors are used because they are very accurate and can capture the required 3D information directly in form of 3D point clouds. However, they are very costly, which is why cameras can pose a cost-effective alternative to them. But, they only capture 2D information (i.e. images), therefore, additional intelligent algorithms are required to overcome this issue.

1.2 Problem Statement

The monocular 3D object detection of vehicles from the ego perspective is the issue solved in this thesis. That means only a single image from a single camera is used. However, there are additional constraints that simplify the task: Only straight highway segments of the *Providentia++* test stretch with precisely calibrated cameras are discussed.

The main challenge of extracting 3D information from 2D images is the lack of depth information. A single pixel on the image plane corresponds to an infinite number of points in 3D space that lie on a straight line. That means, without any additional information, it is not possible to make predictions for vehicles poses. However, due to the fact that vehicles drive on the ground, leveraging the calibration data of the camera, the previously mentioned issue can be compensated. Essentially, an object is on the ground can be located by calculating the intersection of the 3D line that corresponds to the pixel and the ground plane. The size of an object can also be an indicator for its distance to the camera, although, it is not reliable due to varying sizes of vehicles (e.g. a truck far away can have the same size as a closer vehicle on an image). The pose of vehicles also includes the orientation. In 3D space, the can be generally described with three degrees of freedom, though, vehicles on the ground are constrained and can be described with only one (i.e rotation around the normal vector of the ground plane). Because only straight highway segments are contained in the *Providentia++* test stretch, the orientation of a vehicle is also predetermined by the side of the road it is driving on. Technically, the orientation of vehicle changing lines is slightly different from the road direction, however, this small deviation is negligible. All these facts can be combined to estimate 3D bounding boxes with sufficient accuracy.

Also, this entire method should run as part of the *Providentia++* toolchain. This entails that the entire computation (from an input image to 3D bounding boxes) is required to run in real-time. Hence, the number of frames per second (*FPS*), which can be handled continuously, should be larger than 30. This ensures that a live video from the test stretch can be processed without missing any frames. The detected 3D bounding boxes are then further processed in the toolchain, mainly in the tracking and data fusion steps. Lastly, the *digital twin* is created.

1.3 Contribution

My contributions in this thesis are the following:

- The evaluation of two different instance segmentation models for detecting vehicles on a highway within the *Providentia++* test stretch
- A method to estimate 3D bounding boxes for vehicles with different categories on a straight road from only a single 2D image and additional calibration data of the camera
- An evaluation of this method based on the *A9-Dataset* created as part of the *Providentia++* project
- The real-time implementation of this method

- The integration into the existing *Providentia++* toolchain

First, important concepts, terms, and datasets are explained. Next, related work in the field of monocular 3D object detection is presented. Then, my approach is described in detail. Afterward, the approach is evaluated on the *A9-Dataset*. After that, limitations and possible improvements of my work in the future are presented. In the conclusion, the general results and findings are discussed.

Chapter 2

Background

This chapter deals with practical and theoretical background information, which is used in the following chapters. First, a general description of instance segmentation and popular datasets for it is given. Next, 3D object detection of vehicles is presented, also including prominent datasets for it. Then, the model for transformations between 2D image and 3D space is explained. Following, applicable metrics for instance segmentation and object detection are described. After that, general principles of the *Robot Operating System ROS* and its advantages are highlighted. Lastly, the *A9-Dataset* and the structure of the *Providentia++* test stretch are discussed.

2.1 Instance Segmentation

Instance segmentation is the task of detecting multiple objects in an image and their instance masks. Typical models also output a 2D bounding box, a class label of the object, and a confidence score. It is a very active field of research and in recent years, many promising approaches were developed. This also entailed the creation of standardized datasets for producing comparable evaluation results.

2.1.1 COCO Dataset

The most popular one, *Common Object in Context* (*COCO*) [Lin+15], which is also used for standard object detection, features over 123,000 images with more than 886,000 instances from 80 object categories in its most recent version[1]. The categories are tailored for a diverse set of objects in their common environments, however, models trained on it can still be useful for detecting road vehicles due to the included categories: *CAR*, *TRUCK*, and *BUS*.

2.1.2 Cityscapes Dataset

Another large-scale dataset is *Cityscapes* [Cor+16], which was created for image segmentation and instance segmentation on images from urban environments. It contains 5,000 images with fine annotations and 20,000 with coarse ones from 50 German cities featuring objects from 30 classes. Eight of these are vehicles, which makes the dataset potentially very applicable to the task of this thesis.

[1] https://cocodataset.org/#explore

2.2 3D Object Detection of Vehicles

The task of detecting 3D bounding boxes (location, length, width, and height) for vehicles in the 3D world relative to a specified coordinate frame based on multiple kinds of sensor data, even combinations of them, is a very prominent field of research, especially in the last years with the increasing interest in self-driving cars. However, autonomous vehicles (*AVs*) directly are not the only possible application for it. For instance, intelligent infrastructure systems can also benefit from it. The difference between these two tasks is mainly the perspective: one is from inside the vehicle, the ego perspective, while the other one (road-side perspective) is from a fixed point of view without a relation to a specific vehicle. The latter one is relevant to this thesis. Similar to instance segmentation, there are standard datasets for evaluating approaches.

2.2.1 KITTI Dataset

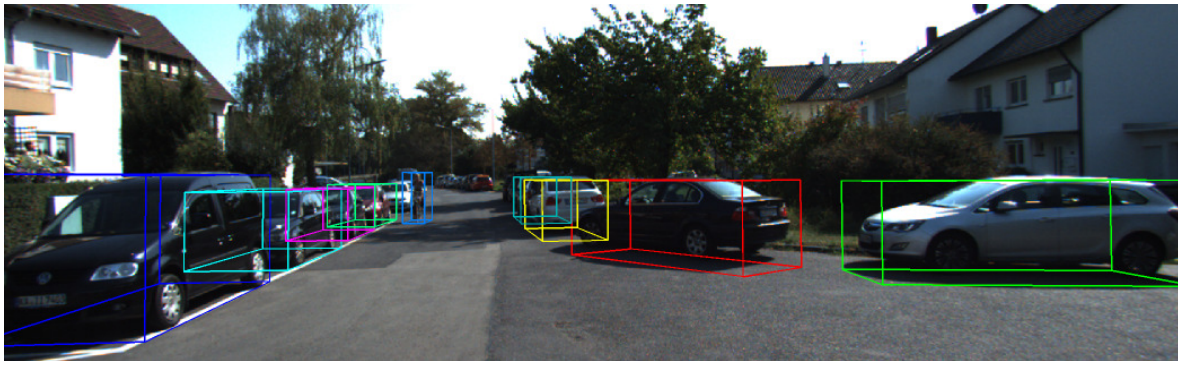


Figure 2.1: An example image from the *KITTI* dataset [GLU12].

The most popular multi-modal dataset, *KITTI* [GLU12], features videos recorded from a vehicle with additional *LiDAR* and *GPS* data. Multiple benchmarks for different tasks are provided by the authors, including 2D and 3D object detection, depth estimation, segmentation, and more[2]. The 3D object detection task is composed of approximately 7,500 training and test stereo images with additional point clouds. In total, they include over 80,000 labeled 3D objects. An example frame is shown in Figure 2.1.

Note, the *Cityscapes* dataset also contains a 3D object detection task[3].

2.2.2 Large-Scale Datasets

While *KITTI* was the pioneering dataset in this field, the focus shifted to a larger scale in recent years. An impressive dataset is the *Waymo Open* dataset [Sun+20], which contains 1,150 scenes, each 20 seconds long. It consists of camera and *LiDAR* data, annotated with 2D and 3D bounding boxes respectively. The goal of it is to bridge the gap between research and real-world use. A similar dataset, called *nuScenes* [Cae+20], also has *radar* and images with 360-degree views. It contains 1,000 scenes, also 20 seconds long, with 23 classes.

This is not an exhaustive list, nonetheless, all the mentioned datasets are focused on the ego perspective. To my knowledge, there is no dataset tailored to the road-side perspective

[2] http://www.cvlibs.net/datasets/kitti/index.php
[3] https://www.cityscapes-dataset.com/benchmarks/#3d-vehicle-detection-task

with similar scale and popularity. However, this does not mean that they cannot be potentially helpful for it.

2.3 Pinhole Camera Model

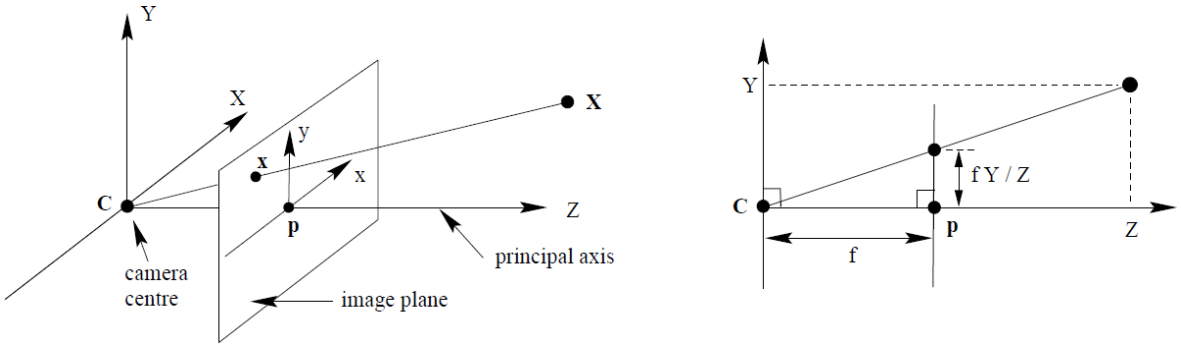


Figure 2.2: The geometric structure of the *Pinhole Camera Model*. It is used to project coordinates from the 3D world into the 2D image plane. (https://hedivision.github.io/Pinhole.html)

Transforming coordinates from the 3D world onto an image can be explained with the *Pinhole Camera Model* [HZ04], which is illustrated in Figure 2.3. The camera center C is the origin of the camera coordinate frame, in which the image plane is orthogonal to the z-axis and has a distance of f to the origin. Now, every 3D point in the camera coordinate system $(x_c, y_c, z_c)^T$ corresponds to a 2D point on the image plane as follows:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \longrightarrow \begin{bmatrix} \frac{f\,x_c}{z_c} \\ \frac{f\,y_c}{z_c} \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \end{bmatrix} \tag{2.1}$$

where x_p and y_p are the pixel coordinates. This transformation can also be expressed with a linear matrix multiplication:

$$\begin{bmatrix} x_p z \\ y_p z \\ z \end{bmatrix} = \begin{bmatrix} f\,x_c \\ f\,y_c \\ z_c \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \tag{2.2}$$

To get the pixel coordinates, the x and y values are divided by z. This can also be described with the intrinsic matrix K, which includes more parameters to compromise the imperfect internal geometry of the camera. It is defined like this:

$$K = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Together with a homogeneous transformation matrix, composed of a rotation matrix R and a translation vector t with respect to the world coordinate system, called the extrinsic matrix, a 3D point in world coordinates $(x_w, y_w, z_w)^T$ can be projected to image coordinates with the following equation. Note, the two matrices together, are also called the *projection* matrix.

$$\begin{bmatrix} x_p z \\ y_p z \\ z \end{bmatrix} = K \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{2.4}$$

Due to the projection from 3D to 2D, information is lost, however, a single point on the image plane corresponds to a line in the 3D camera coordinate system, which spans from the origin to the image coordinates. This allows projecting a point from the image to a 3D point on the ground, e.g. the road, by computing the intersection of the line and the ground plane. Assuming the ground plane is defined as $z = 0$, the transformation can be split into simple steps:

1. Re-project the pixel coordinates into the 3D camera coordinate system using K^{-1}
2. Rotate the points back into the world rotation with R^{-1} ($R^{-1} = R^T$)
3. Divide the new x and y values by z
4. Multiply with the negative height of the camera
5. Add the translation vector of the camera position

Afterward, the z value is zero and the x and y coordinates correspond to the location on the ground.

2.4 Metrics

2.4.1 Binary Classification

For a binary classification with N number of samples and labels, the output of a model can be partitioned into four categories, as shown in Table 2.1. This is generally known as the confusion matrix.

	Output True	Output False
Label True	True Positive (*TP*)	False Negative (*FN*)
Label False	False Positive (*FP*)	True Negative (*TN*)

Table 2.1: Confusion matrix for binary classification.

These can then be combined to form additional metrics with more complex meanings. The most common metric is *Accuracy*, it is defined as:

$$accuracy = \frac{TP + TN}{N} \tag{2.5}$$

It describes the percentage of correctly classified examples. However, this loses meaning when the number of samples per class is imbalanced. For example, a trivial model, that always predicts *True*, would achieve an accuracy of 95 % on a dataset with 95 *True* samples and 5 *False* samples. To assess such a situation better, different metrics are needed. Two popular ones are *Precision* and *Recall*:

$$precision = \frac{TP}{FP + TP} \tag{2.6}$$

$$recall = \frac{TP}{TP + FN} \tag{2.7}$$

Precision gives a value on how many samples, that are predicted as *True*, are actually *True*. In contrast, *Recall* describes how many samples, that are actually *True*, are predicted as *True*. Both can also be combined:

$$f1 = 2 * \frac{recall * precision}{recall + precision} \tag{2.8}$$

The *F1-Score* is the harmonic mean of *Precision* and *Recall* and is a great indicator of overall model performance.

2.4.2 Object Detection

The previously explained metrics can also be used for object detection, though, detections and labels have to be associated with each other first. This is often done by calculating the ratio of intersection and union for the 2D bounding boxes (*IoU*). Detections and labels are matched if they have an *IoU* above a specified threshold value (e.g. 0.7). Then, the number of detected objects is *TP*, the number of undetected objects is *FN*, and the number of invalid detections is *FP*. The notion of *TN* is not defined here, however, *Precision*, *Recall*, and *F1-Score* do not require *TN* anyway.

There are two main ways to compute these metrics on an entire dataset of images. One is called *Macro-Average*, first, the metrics are computed per image individually and then averaged over all of the images. The other one is called *Micro-Average* and works the other way around, *TP*, *FP*, and *FN* are computed for each image and then combined into three values for the entire dataset. Then the metric is computed with three dataset-wide values.

In the domain of object detection, there exists another popular metric called *AP*, which is the area under the curve of *Precision* per *Recall* for all *IoU* threshold values. This is normally approximated with e.g. 11 fixed *IoU* values. Technically, this metric is computed per class, and averaging them is called *mAP*, however, sometimes there is no distinction between them (e.g. in the *COCO* dataset *mAP* is called *AP*).

2.4.3 3D Object detection

For 3D object detection, the metrics based on binary classification are not sufficient to evaluate the performance of a model. This is because the error of the 3D bounding box is also very important. One way to do this is called *Mean Absolute Error* (*MAE*). It can be calculated element-wise (i.e. per x-location, length, etc. individually) as follows:

$$MAE = \frac{1}{N} * \sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{2.9}$$

where y_i is the actual e.g. length value and $\hat{y}_i$ is the predicted value for a single vehicle. To compute this metric for an entire dataset, *Macro-Average*, where the *MAE* values are computed per image and then averaged, or *Micro-Average*, where a single *MAE* value is calculated over all vehicles from all datasets, can also be used.

There are other metrics, such as *Root Mean Squared Error* (*RMSE*), which work with squared errors instead of absolute values. However, *MAE* can be interpreted as: "How wrong are the values on average?", hence it is very useful for an evaluation that produces an intuitive understanding of the error values.

2.5 Robot Operating System

The *Robot Operating System* (*ROS*)[4] is an open-source middleware software framework for robot applications. However, it is not a real operating system and the term robot can be widely interpreted.

Its main intent is to provide a standard way of combining multiple processes, that work together on a task, and their communication between each other. This should not introduce much overhead and act as a thin layer above the original source code. A single process is called *Node* and it can communicate with other processes using messages. This is done with *Topics*, where each *Node* can send messages to a chosen *Topic* and another *Node* can subscribe (i.e. listen to it) and receive the messages. The form and type of each message are also specified.

Software is partitioned into *Packages* and can be written in multiple programming languages. Multiple *Packages* can also be combined. Additionally, multiple tools and software packages are available to facilitate a multitude of tasks, irrelevant to the thesis.

2.6 Providentia++ A9-Dataset

Figure 2.3: An overview of the *Providentia++* test stretch from [Cre+22].

The *A9-Dataset* [Cre+22] is a multi-modal dataset recorded on the *Providentia++* test stretch near Munich. It consists of 1098 labeled frames, while 642 of those frames were recorded with cameras, the remaining part was captured using *LiDAR* sensors. In the following, only the camera part of the *A9-Dataset* is discussed.

The images were captured from the two measurement points *S40* and *S50*, which are displayed in Figure 2.3. The distance between *S40* and *S50* is less than 500 meters. There are two cameras at each measurement point. One camera has a narrow field of view with a focal length of 50 mm and the other camera has a wider field of view with a focal length of 16 mm. The cameras at both measurement points face the same road segment from opposite sides. In Figure 2.4, an example image for each camera is shown.

All images contain a total number of 11,353 labeled 3D objects from nine different categories. The categories and number of labels per category are displayed in Table 2.2. The

[4] https://www.ros.org/

Category	#Labels
CAR	8156
VAN	1174
TRAILER	1048
TRUCK	853
BUS	59
MOTORCYCLE	33
PEDESTRIAN	20
SPECIAL_VEHICLE	9
BICYCLE	1

Table 2.2: Number of labels per category in the camera part of the *A9-Dataset*.

majority of the vehicles are cars, trucks, trailers, and vans. In Table 2.3, the number of frames and vehicles are grouped per camera.

Camera	#Frames	#Labels	Avg. #Labels per Frame
S40 North 50 mm	179	5484	30.64
S40 North 16 mm	174	2109	12.12
S50 South 50 mm	174	2848	16.37
S50 South 16 mm	114	912	8.00

Table 2.3: Number of frames and labels per camera in the *A9-Dataset*.

The labels for each frame are saved as *JSON* files, which contain the 3D locations and dimensions for all labeled vehicles. For each camera, calibration data (i.e. intrinsic and extrinsic camera matrices) is also provided.

(a) S40 North 16 mm

(b) S40 North 50 mm

(c) S50 South 16 mm

(d) S50 South 50 mm

Figure 2.4: Example images from all four cameras in the *A9-Dataset*.

Chapter 3

Related Work

In this chapter, related work is presented, which either supports the proposed method or aims to solve the same or a similar task. First, two different instance segmentation models are presented. Then, different methods for 3D object detection of vehicles are addressed.

3.1 Instance Segmentation

There is an abundance of different architectures for instance segmentation available, often the implementation is also provided by researchers. These models are evaluated and compared on standard datasets like *COCO* or *Cityscapes*, as presented in Subsections 2.1.1 and 2.1.2. Additionally to the implementation, pre-trained weights based on these datasets are also frequently provided. In the following two different approaches are described. Note, there are even more sophisticated architectures with better performance, that make use of *Transformers* e.g. [Che+21], yet, they neither perform real-time nor is a more detailed mask required for the approach presented in this thesis. The latter is explained later in Subsection 4.2.

3.1.1 Mask R-CNN

One popular model is called *Mask R-CNN* [He+18], is an extension of a model for object detection called *Faster R-CNN* [Gir15] and consists of two stages. First, a *Region Proposal Network* (*RPN*), which includes a *backbone* for feature extraction from an image, proposes 2D bounding boxes for potential objects. Second, three *heads* use them for producing instance masks, categories, and refined 2D bounding boxes. Multiple variants with different *backbones* are available, implemented with *Detectron2* [Wu+19], including pre-trained weights on *COCO* and *Cityscapes* as part of the their *Model Zoo*[1]. It helped to experiment and develop the approach because of its excellent usability, though, it is not practical for real-time use due to its considerable inference time, even with the smallest *backbone*, as later presented in Subsection 5.1.1.

3.1.2 YolactEdge

A different model for instance segmentation, *YolactEdge* [Liu+21a], also known as just *YOLACT*, aims to explicitly solve the real-time issue. Moreover, it accomplishes this task for images

[1] https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md

(with a resolution of 550x550) on *edge* devices (i.e. computers close to their use case and generally with less computation power). It leverages *TensorRT*[2], which greatly improves inference time with certain optimizations. This also enables real-time inference for images with higher resolution on regular devices with a descent *GPU*. In contrast to *Mask R-CNN*, which is an extension of the two-stage model *Faster R-CNN*, it consists of a single stage and similarly is an extension of a one-stage object detection model like *YOLO* [Red+16]. The single stage performs two parallel tasks at once: one produces general instance masks for the whole image, called *prototype masks*, while the other detects objects (single-stage) and their relation to the *prototype masks*. At last, both results are combined for the finished instance masks. The authors also provide an open source implementation with pre-trained weights[3] on *COCO*, though not on the *Cityscapes* dataset. This model allows the proposed method to run in real-time and is therefore used in the final implementation.

3.2 3D Object Detection of Vehicles

While there is a multitude of approaches, for 3D object detection of vehicles, using other sensors than cameras, this thesis only focuses on images from a single camera (monocular 3D object detection). Therefore, only those approaches, which work based on single images are explored here. This is a particularly interesting case because missing information from 2D to 3D has to be inferred, as described in Section 1.2. Similar to the datasets for monocular 3D object detection, as described in 2.2, the approaches are primarily designed for AVs, i.e. the ego perspective, but that does not mean that they cannot potentially perform on tasks from a road-side perspective. In the following, some approaches from two prominent types are shown. Lastly, methods developed for the road-side perspective are presented.

3.2.1 Keypoint Estimation

A keypoint is a point of interest in the image, which is directly estimated. All of the following methods use this technique to some degree.

A popular approach, which outperformed all other monocular methods on the *KITTI* dataset, is called *SMOKE*. It is a single-stage detector, which estimates the center of the 3D bounding box, projected onto the image, as described in 2.3, as a keypoint and simultaneously regresses variables for the dimensions and orientation of it. Then, both results are combined for a single prediction.

Another method, named *RTM3D* [Li+20], directly estimates nine keypoints, the center and all eight corners of the projected 3D bounding box, on the image plane. Afterward, the keypoints are used to estimate the real 3D bounding box using geometric constraints, such that the reprojection error of the estimated corners, with respect to the 2D corners (i.e. the estimated keypoints), is minimized.

In contrast to the two previously mentioned methods, *MonoEF* [Zho+], specifically addresses the problem of extrinsic perturbations due to e.g. potholes or slope of the road. The method also detects the horizon change and vanishing point to predict the extrinsic camera parameters. Otherwise, the approach follows a similar approach like *SMOKE*.

At last, *AutoShape* [Liu+21b] uses another interesting idea and does not share the assumption, that each vehicle is a cuboid. Instead, *CAD* models of vehicles are used to learn meaningful keypoints based on them.

[2] https://developer.nvidia.com/tensorrt
[3] https://github.com/haotian-liu/yolact_edge

There are many more approaches, however, they are all designed for the ego-perspective, which does not make them inherently useless, however, *SMOKE* and *RTM3D* were tested on my task but did not produce satisfying results. In Figure 3.1, the results of the *RTM3D* on an image of the *Providentia++* test stretch are presented. Generally, these methods often lack the ability to work with full-height images.

Figure 3.1: Results of the *RTM3D* model on an image of the *Providentia++* test stretch.

3.2.2 Depth Estimation

Another family of approaches tries to predict the missing depth information directly and infer vehicle poses from that. Wang et al. [Wan+18] propose a method to create image-based depth maps and convert them to *pseudo-LiDAR* representations. These mimic the output of regular *LiDAR* sensors, i.e. 3D point clouds. Therefore, existing 3D object detection methods for point points clouds can be used. Contrary, Park et al. [Par+21] argue that end-to-end methods perform better. They propose single-stage model, called *DD3D*, that achieves state-of-the-art results on the datasets *KITTI* and *nuScenes*.

Both of these approaches are focused on the ego perspective and do not make use of the fixed camera position and the straight highway segments. They were not tested on the vast range of depth in the *A9-Dataset*.

3.2.3 Road-Side Approaches

Guo et al. proposed a method [Guo+21], which is explicitly designed for road-side perspectives, nonetheless, it can also be employed for ego perspectives. They use a pre-trained instance segmentation model that produces an instance mask for each vehicle in the image as backbone. Then, two lines are fitted to the bottom contour of the mask, using *K-Means* clustering with $K = 2$ and lines instead of centroids. With these, three points, which are furthest away and should resemble wheels, are selected as the corners of the 3D bounding box. Unfortunately, this step is not clearly explained, especially how to infer three points from two lines and how the situation, when these do not enclose a 90 degree angle on the ground plane, is handled. Also, the assumption, that these points are the wheels is not validated. Next, the vehicle dimensions and location are estimated from the three corners, also using the *Pinhole Camera Model* to project points from the image onto the ground, as explained in Section 2.3. At last, they introduce a post-processing step, using a *Maximum a Posteriori Estimation* (*MAP*), to improve the 3D bounding box based on the 2D bounding box produced by the instance segmentation model. An implementation is not provided, therefore it is not clear, how the authors handled the mentioned issues. However, the idea to use the bottom

contour of instance masks as an approximation of the edges of the vehicle, that contact the ground, is the main inspiration for the method proposed in this thesis.

Another compelling approach, published by Clausse et al. [CBL19], was designed to extract vehicle trajectories from videos for creating a dataset. For each frame, vehicle poses are detected and then later combined. The detection part also uses instance segmentation as the first step and then estimates a 3D bounding box on the ground by maximizing the overlap from the mask and the area of a box projected onto the image. A limitation is that pre-defined dimensions are used and not individually estimated per vehicle. The part for the 3D object detection is less complex than other methods, however, this is due to the focus of tracking. There, estimates can be greatly improved by combining information from the same vehicle in multiple frames, e.g. the orientation should be similar to the direction of motion. Because of this, it is not directly applicable.

The *Traffic-Net* [RAM21] approach is designed to be a complete traffic monitoring solution, i.e. vehicle detection, tracking, speed estimation, and congestion detection. This allows, similar to the previously mentioned method, to support the 3D box estimation with information from tracking. For the dimensions, pre-defined values are also used here. This is therefore also not applicable for my task.

Zhu et al. [Zhu+22] published a procedure with a completely different idea. They project the complete image onto the ground plane and detect rectangular 2D bounding boxes there. Nonetheless, they do not predict height values. On the contrary to other methods, they do not rely on intrinsic- and extrinsic camera parameters.

Note, this is not an exhaustive list. However, to my knowledge, there does not exist a method, which focuses on highly calibrated cameras only for highway scenarios (i.e. only two directions with vehicles facing the opposite orientation), which is the case for the *A9-Dataset* created within the *Providentia++* [Cre+22] project. Therefore, the custom approach proposed in this thesis is exactly tailored for this.

Chapter 4

Approach

The proposed approach for estimating 3D bounding boxes from a single image consists of five main steps, shown in Figure 4.1. First, an already pre-trained instance segmentation model is used to detect vehicles, especially their instance masks are important. Then, the bottom contour for each mask is determined and afterward, projected onto the ground plane, based on the exact calibration of the camera. Next, the projected contours are used to estimate 2D bounding boxes on the ground, which are aligned to the direction of the road. This is tailored for the highway scenarios, as contained in the *A9-Dataset*, and therefore heavily relies on the stationary and calibrated camera perspectives. Afterward, the observed mask heights are used to estimate the real height of each vehicle. At last, these estimations are used to correct potentially wrong vehicle categories.

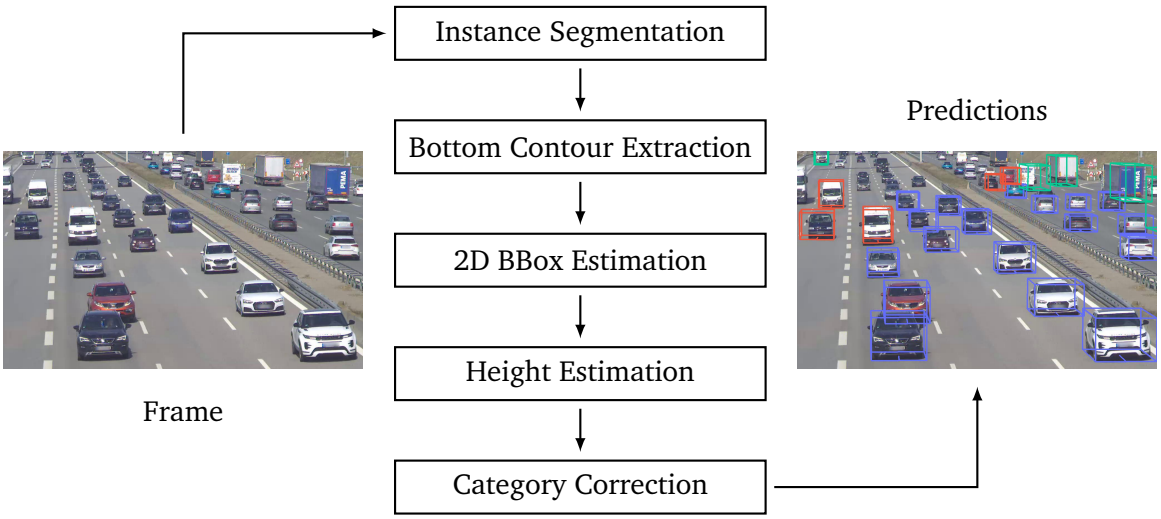


Figure 4.1: The steps of the proposed approach from an original image to final predictions.

The hyperparameters for each step are also described, though, no explicit values are given here because they depend on the chosen model for instance segmentation. In Subsection 5.2.4, they are determined for multiple models as part of the evaluation on the *A9-Dataset*. In the following, each step is explained in detail.

4.1 Instance Segmentation

The backbone of the approach is an instance segmentation model, of which many implementations exist, as described in Section 2.1. It is not required that they are specifically trained

on a dataset for vehicles, nonetheless, it is advantageous if they generalize well enough and can predict more than a single class for vehicles.

For each detection, a binary instance mask, 2D bounding box, class label, and score are produced, which are all used in later steps. Depending on the particular dataset, on which the model was trained, class labels need to be converted to categories used in the *A9-Dataset*, as presented in Section 2.6. Detections with class labels, that do not have a corresponding category, are removed. These depend on the dataset, that was used for training the model (e.g. only classes "car", "truck", "bus" are relevant in the *COCO* dataset). Furthermore, predictions with a score below a specified threshold value are also filtered out.

If specified, instance masks, that are closer to the image edges than a specified threshold value, can also be removed. The reason for that is, that these potentially belong to truncated vehicles, which leads to wrong results because of an incomplete mask, not covering the entire vehicle. Additionally, instance masks with an approximated width, which is the square root of the area (i.e. sum of the binary instance mask), below a specified value can also be deleted. A visualization of produced instance masks and 2D bounding boxes on an image is shown in Figure 4.2.

The hyperparameters summarized for this step are a score threshold, an approximated mask width threshold, a minimum margin to image edges, and a mapping from possible class labels to categories.

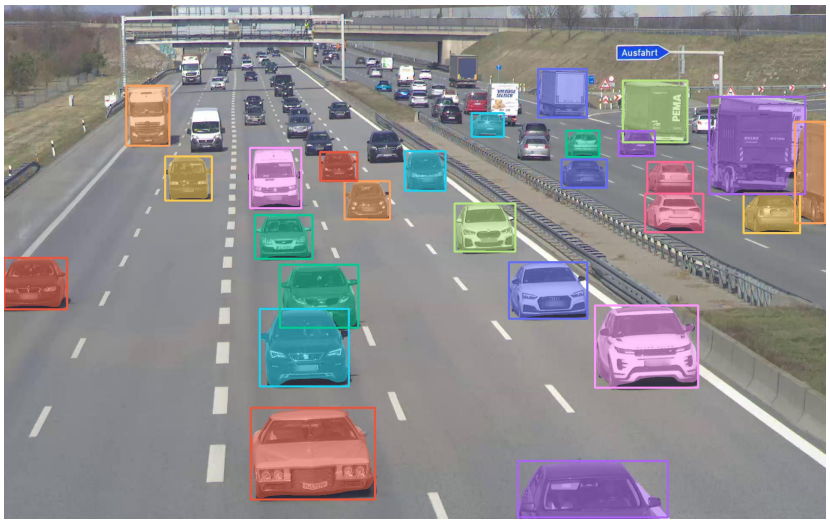
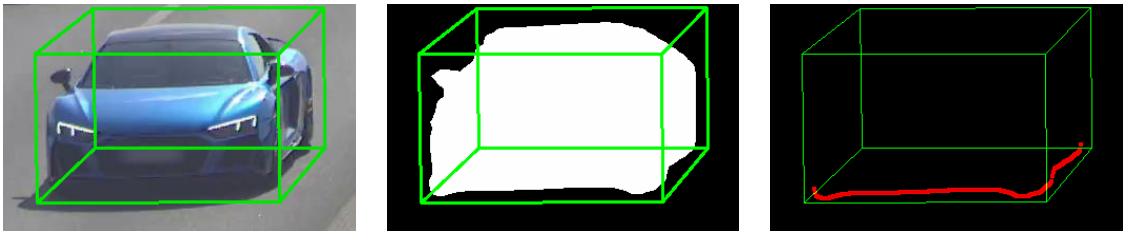


Figure 4.2: Instance segmentation masks on an image including 2D bounding boxes from a *Mask R-CNN* trained on the *COCO* dataset.

4.2 Bottom Mask Contour Extraction

The bottom contour is determined for each instance mask. This is done by iterating over all pixel-columns, where at least one pixel is part of the mask, and selecting the lowest one. An illustration of a vehicle, the corresponding mask, and the resulting bottom contour is shown in Figure 4.3.

The reasoning for using this contour is the following. Each vehicle is approximated as a cuboid on the ground. That means that there are four edges contacting the ground, which together form the 2D outline of the vehicle, including length and width. With the exception of five viewing angles (facing exactly one of the sides or the roof), there are always two of the edges on the ground visible. The bottom contour of the instance mask of such a



(a) *Image* **(b)** *Instance Mask* **(c)** *Bottom Contour*

Figure 4.3: Image, instance mask, and bottom contour of a vehicle.

perfect cuboid, projected onto the ground plane, as explained in Section 2.3, would exactly represent these two edges, which are theoretically sufficient to infer the 2D box, including location and dimensions. However, vehicles are not ideal cuboids and instance masks are not perfect. Nonetheless, the contour should resemble an L-shape, when projected onto the ground. Based on this, the orientation, location, and dimensions can be estimated.

Due to ground clearance of vehicles and poor mask quality, the instance masks can be a bit too high, as visible in Figure 5.1. This can be corrected by shifting all contours a fixed number of pixels down, which is the only hyperparameter of this step.

4.3 2D Bounding Box Estimation

The bottom contours from an image of an intersection resemble an L-shape, as shown in Figure 6.2. However, for those on the highway, as part of the *A9-Dataset*, it is not that clear because only a small portion of a side edge is visible for many vehicles. Furthermore, small errors of the masks are amplified due to the projection on the ground. Though, all camera perspectives in the A9 dataset observe straight highway segments, as shown in Figure 2.3. Therefore, the orientation of the perceived vehicles is fixed (modulo 180 degrees). This information should be used, i.e. the orientation of the L-shape is known. Therefore, only the dimensions and the location have to be estimated.

The direction of the road can be specified as a 2D hyperplane in the road coordinate frame. On the positive side of it, the vehicles point to the left and on the other side the other way. This allows to fully specify the correct orientations for a highway segment with opposing lanes. To simplify the next calculations, the ground contours and camera positions are rotated into a coordinate frame with the x-axis along the hyperplane (road direction and car lengths). The y-axis is aligned with the normal vector (car widths). The orientation can now be inferred using the sign of the y coordinate. An example of the 2D hyperplane and the orientation and direction of vehicles is displayed in Figure 4.4.

In the rotated frame, a naive bounding box can be constructed by calculating the minimum and maximum values for x and y coordinates of the ground contour. However, due to the missing side and errors of the contour, this leads to wrong and enlarged bounding boxes. Note, as shown in 4.5, the corner nearest to the camera position is most accurate. Intuitively, this makes sense, since the corner is definitely very close to the ground and completely visible (if not occluded or truncated). To correct the enlarged length and height, the location of the corner is fixed and length and width values are corrected according to maximum and minimum values based on the category of the detection. If one dimension is clearly not visible (e.g. length of a car is estimated as 1 m), the mean length value for the category is selected. This procedure leads to fitting 2D dimensions for the detected vehicles as shown in 4.5. From the fixed corner, width, and length, the center location can be calcu-

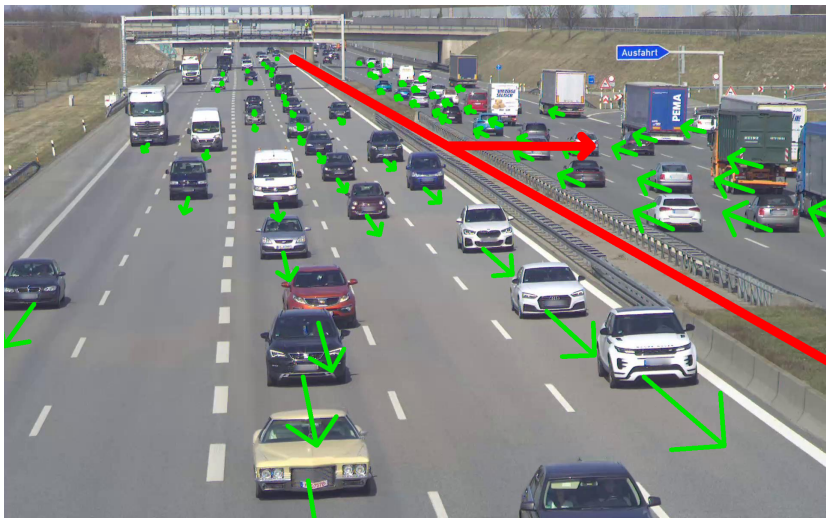


Figure 4.4: 2D hyperplane, including the normal vector, in red and orientation of the vehicles in green. On the positive side of the 2D hyperplane, the vehicles drive to the left relative to the normal vector.

lated. The maximum, minimum, and mean values per dimension for each vehicle category are additional hyperparameters for this step. In Subsection 5.2.4 in the evaluation, they are computed based on all labeled vehicles in a subset of the *A9-Dataset*.

4.4 Height Estimation

The ground contour is by design not useful for estimating the height of the vehicles. A simple improvement would be to use the detection category and set height based on per-category means. Still, using the constraint, that the camera view is roughly aligned with the road direction in highway scenarios, the mask height can be used to estimate the height. If a vehicle is far enough away and only the front or back is visible, not the roof, the pixel height is inverse-proportional to the distance to the camera. This relationship is defined by the focal length of the camera i.e. $d * m = f_y$ where d is the distance to the camera, m is the observed pixel height, and f_y is the focal length.

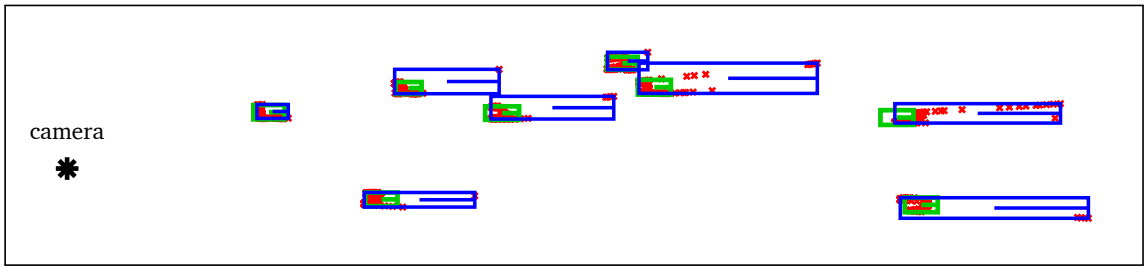
Though, the observed mask height is not the real height, especially for vehicles closer to the camera where the roof is visible. The fact, that the distance from the camera and camera height is known, can be used to estimate the real height with the Equations 4.1 and 4.2, where z_c is the height of the camera, l_v the mean length, and h_v the mean width for the vehicle category. It is derived from a simplified geometric configuration illustrated in 4.6.

$$\alpha = \arctan(\frac{z_c}{d}) \tag{4.1}$$

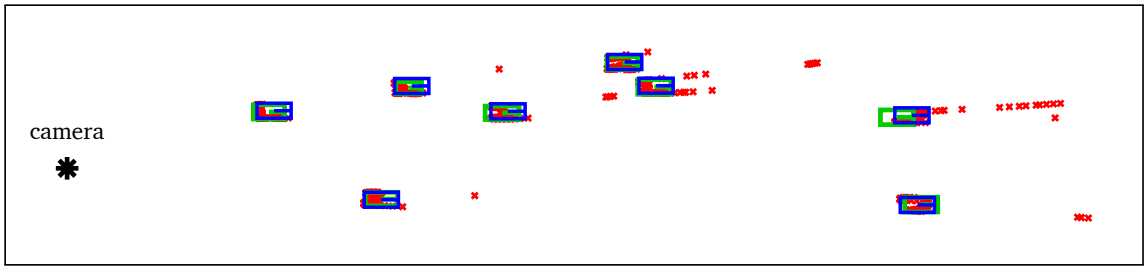
$$h = \frac{d * m}{f_y * (\cos(\alpha) + \frac{l_v}{h_v} * \sin(\alpha))} \tag{4.2}$$

4.5 Category Correction

The estimated heights are then used to correct potentially wrong vehicle categories based on the computed maximum and minimum height values, as mentioned in Subsection 4.3.



(a) *Naive*



(b) *Improved*

Figure 4.5: Comparison of naive- and improved 2D bounding boxes (blue) calculated from projected bottom contours (red) compared to labels (green). The nearest corner of the naive bounding boxes to the camera is the most accurate. The improved boxes share this corner but length and width are adjusted.

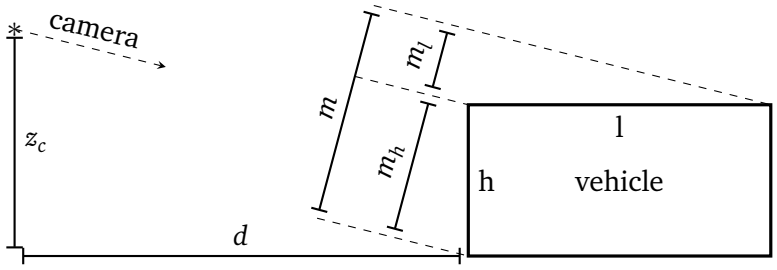


Figure 4.6: The geometric configuration of a camera and the vehicle, which is used for estimating the height. The observed height of the instance mask of the vehicle is composed of the front/back and the roof. The exact ratio depends on the vehicle dimensions and the camera angle.

However, first, maximum and minimum values of the categories *CAR*, *VAN*, and *TRUCK/BUS* are aligned, such that they do not overlap. This is done by setting the maximum height for cars and the minimum height for vans to the average of their mean heights. The same is done for vans and trucks/busses. Trucks and busses are different categories, though, they share the same minimum value for consistency. With that, the categories of vehicles can be corrected, based on the non-overlapping height regions. Note, the *COCO* dataset does not have a *VAN* class, however, with this method it can still be inferred.

The assumption, that vehicle heights can be roughly partitioned, based on the categories, is supported by Figure 4.7, where the distributions of height values per category for all vehicles in the *A9-Dataset* are shown. It is clearly visible, that the height range can be partitioned into non-overlapping segments for each category, where the majority of all vehicles can be correctly classified based on their height values. This is also discussed in Subsection 5.2.5, as part of the evaluation.

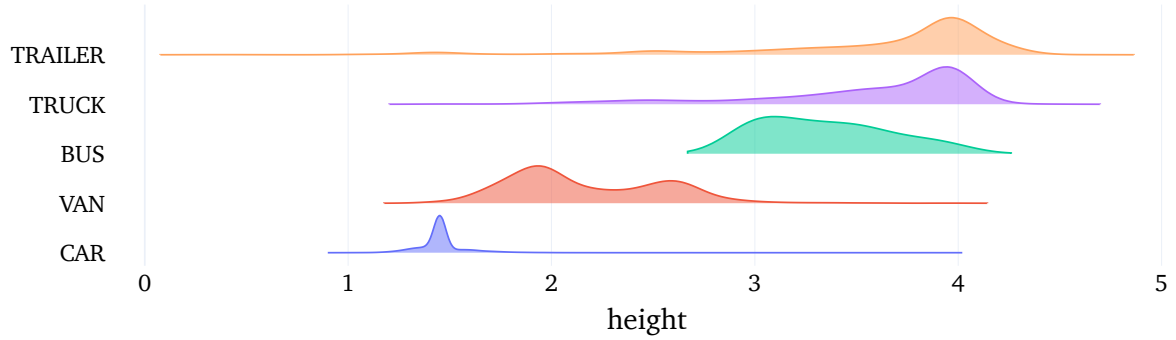


Figure 4.7: Distribution of vehicle heights grouped by category for the *A9-Dataset*. It is clearly visible, that the height range can be partitioned into non-overlapping segments for each category, where the majority of all vehicles can be correctly classified based on their height values.

4.6 Providentia++ Toolchain Integration

The approach is also integrated into the *Providentia++* toolchain, which runs with *ROS*. As mentioned in Section 2.5, one of the core principles of *ROS* is to only act as a thin layer on top of the original software. That means, only two interfaces, to parse images from messages and to publish the 3D bounding boxes from predictions, have to be implemented. With the latter, predictions from multiple frames can be combined in the tracking part and eventually merged with additional data to construct the *digital twin*.

Note, an important aspect to it is that the entire approach is computed inside a single node and not split into instance segmentation and bounding box estimation. Adhering to good software engineering principles, they should be separated, to make testing and maintenance easier, however, serializing the instance masks into a message is infeasible with real-time constraints because it includes transferring tensors from the *GPU* to *CPU*. This entails that the calculation of the bottom contour of the instance masks is also computed on the *GPU* and only then are the points transferred to the *CPU* for the remaining steps of the approach.

Chapter 5

Evaluation

In this chapter, the two parts of the approach are evaluated. First, two instance segmentation models and their variants are compared on accuracy and inference time. Second, the proposed approach is tested with these models and evaluated on the *A9-Dataset*. All results are reproducible with the *Jupyter* notebooks and *Python* scripts as part of the submitted code.

5.1 Instance Segmentation

For the first part of my approach, the instance segmentation, two different models are evaluated. First, *Mask R-CNN* as part of the *Detectron2* because it is very easy to use and was therefore crucial for developing and experimentation. Second, *YOLACT*, since it offers real-time capabilities in its *YolactEdge* variant which is needed for using it as part of the *Providentia++* toolchain. For both models, pre-trained weights on *COCO* with different backbones are provided by the authors, as discussed in Subsections 3.1.1 and 3.1.2. They are also evaluated with respect to their mask *mAP* on the *COCO* dataset by the authors. As mentioned in Subsection 2.1.1, it is a general-purpose dataset but it also generalizes well enough to suffice in this use case, which is shown in Figure 5.6.

In Table 5.1 you can see the eight different evaluated models with their *mAP* values. There is one exception, the last model was trained and evaluated with *Cityscapes*. You can clearly see that the *Mask R-CNN* outperforms the other architecture on *mAP* but the frames per seconds *FPS* are substantially lower. The *YolactEdge* variants run both at 60 *FPS* and are therefore the only ones with real-time capability (i.e. FPS $>=$ 30). With 60 *FPS*, which is equal to 16.7 ms per frame, there is a buffer of another 16.7 ms for the remaining part of the approach. Their mask *mAP* is much lower, however, as shown in Figure 5.1, it still suffices for my task. The procedure for measuring the FPS is explained next.

5.1.1 Inference Time

All models were benchmarked on the 642 camera frames of the *A9-Dataset*. The *YolactEdge* variants use *TensorRT* with the *FP16-only* configuration and fast *NMS*. The evaluation ran on an *Ubuntu* 20.04 server with an *NVIDIA GeForce RTX 3090 GPU* and *AMD EPYC 7282 CPU*. The measured time includes transferring the original frames from *CPU* to *GPU* since it should resemble the actual use case. Note, the *FPS* results shown in Table 5.1 are not comparable to benchmark times by the authors, since different hardware and image resolution is used by them (e.g. *YolactEdge* was benchmarked with 550x550 resolution on an *NVIDIA GeForce RTX 2080 Ti*).

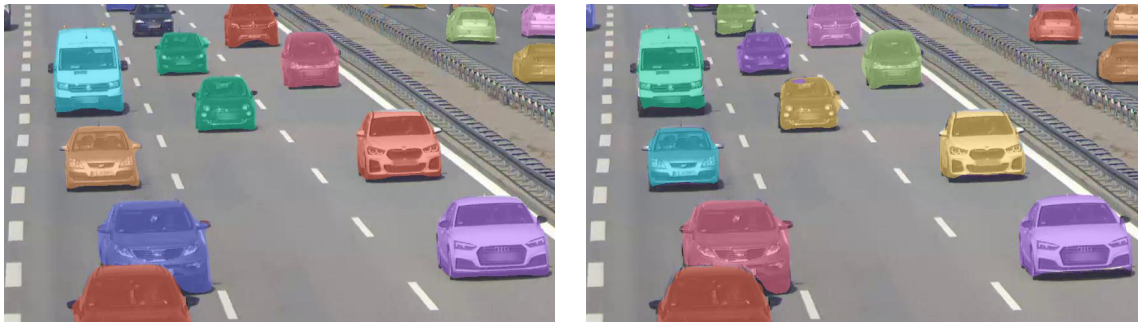
Model	Backbone	Mask mAP ↑	FPS ↑
Mask R-CNN	*R-50-FPN*	37.2	14.5
Mask R-CNN	*R-50-FPN**	36.5*	11.6
Mask R-CNN	*X-101-FPN*	39.5	9.2
Mask R-CNN	*X-152*	44.0	4.5
YOLACT	*R-50-FPN*	28.2	30.2
YOLACT	*R-101-FPN*	29.8	22.6
YolactEdge	*R-50-FPN*	27.0	**59.7**
YolactEdge	*R-101-FPN*	29.5	**59.0**

Table 5.1: Comparison of pre-trained (*COCO*, *Cityscapes**) *Mask R-CNN* and *YOLACT* models. The *mAP* values refer to the validation splits of the respective datasets, on which the models were trained on. The *YolactEdge* variants are the only ones with sufficient *FPS* for real-time use.

5.1.2 Qualitative Results

Figure 5.6 shows the detected instance masks and 2D bounding boxes, with a score above 0.7, generated by the eight model variant on a single frame of the *A9-Dataset*. Nearly all vehicles, within a reasonable distance to the camera, are detected. This is just a single frame and is only intended as a visual example to give a general understanding of the results (e.g. the white van is generally detected with a high score by the *Mask R-CNN R-50-FPN*).

The quality of the instance masks is slightly worse for the best *YolactEdge* model in comparison to the worst *Mask R-CNN* model, as shown in detail in Figure 5.1. Though, the difference of the bottom contour, which is essential for my approach, is not as substantial as the run-time difference. This is also compromised by shifting down the contour a small number of pixels, as mentioned in 4.2.



(a) *Mask R-CNN R-50-FPN* **(b)** *YolactEdge R-101-FPN*

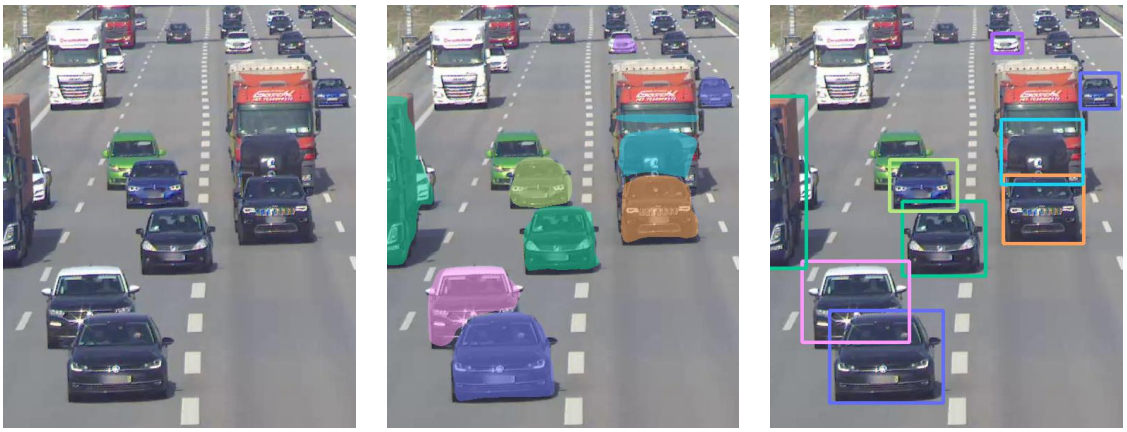
Figure 5.1: Comparison of mask quality between pre-trained (*COCO*) *Mask R-CNN R-50-FPN* and *YolactEdge R-101-FPN* models. The masks of *YolactEdge* are slightly worse but the bottom contour does not differ much. A video for masks generated by the *YolactEdge* is also available (https://youtu.be/2F2bnz7Mtis).

5.1.3 Limitations

One limitation is that *YolactEdge* is only pre-trained with *COCO*, which only has three valuable classes (*CAR*, *BUS*, and *TRUCK*). Note, there are pre-trained weights for an older version of *YolactEdge* on the *Cityscapes* dataset available from a third party, though, to remain comparable, they are not considered here because they are not provided by the authors. Nonetheless,

all relevant vehicles are detected but it cannot be differentiated between all categories in the *A9-Dataset*.

Another important limitation is that due to the relatively low camera position, vehicles can be partially or fully occluded as shown in Figure 5.2. There, the truck (light blue) is covered by a small trailer of a vehicle but the mask is nearly correct because the truck is slightly wider. However, for the two partially occluded vehicles (pink and orange), this is not the case. Vehicles, which are truncated because of the image edges, also produce incomplete instance masks. Both of these lead to invalid bottom contours, which are not intentionally handled by my approach.



(a) *Original image* **(b)** *Instance masks* **(c)** *2D bounding boxes*

Figure 5.2: Instance masks and 2D bounding boxes for partially and fully occluded vehicles. The mask of the truck (light blue) is intentionally drawn on top of the trailer (red) to visualize its full extent.

Also, cameras are heavily influenced by the environment. At night or with bad lighting conditions, many vehicles are not detected. Additionally, bad weather, especially rain or snow, can decrease the image quality. These problems are inherent to camera images and cannot be fixed easily. In Figure 5.3, instance masks generated by a *YolactEdge R-101-FPN* model from an image with rainy weather and unfavorable lighting conditions are shown. The masks are deformed and enlarged due to the blur of the headlights and more vehicles, especially those far away with bright headlights, are not detected.

Figure 5.3: Instance masks generated by a *YolactEdge R-101-FPN* model on an image with rainy weather and unfavorable lighting conditions. A video is also available (https://youtu.be/E0yQkHH5vK0).

5.2 3D Bounding Box Estimation

In this section, the proposed approach with each instance segmentation model is evaluated on the camera images in the *A9-Dataset*. First, characteristics of the dataset, which influence the results are discussed. Second, the used metrics and the procedure to compare predictions and labels are explained. Then, the optimal hyperparameters for each instance segmentation model are computed, based on a quarter of the dataset. At last, the results are evaluated and analyzed on the remaining three quarters. Note, normally the training set is way larger but here one quarter is sufficient due to the small number of hyperparameters and no "real" training. This also allows for a better evaluation of a larger test set.

Note, only models pre-trained on the *COCO* dataset are discussed here because there are no weights directly for *YolactEdge* on *Cityscapes* available without converting them from *YOLACT*, as discussed in Subsection 5.1.3.

5.2.1 A9-Dataset

The 3D bounding boxes were hand-labeled on the image and therefore contain ground truth errors. This is normal for datasets, however, a small number of pixels in the image can turn into multiple meters when projected on the ground, especially if the distance between the labeled vehicle and the camera is large. Also, trucks and their trailers are labeled as two separate vehicles. This is not accounted for in the approach and leads to larger errors for these categories.

Another aspect is that there are unlabeled vehicles on many frames, which are too far away for correct labeling. This leads to the issue of correctly counting false positives. Since the two 50 mm cameras face each other and there are less than 500 meters apart, it is a reasonable thought to only evaluate vehicles less than 250 meters away respectively. For the 16 mm cameras, which cover a smaller area, 125 m is similar with respect to vehicle size at the cutoff distance. Therefore, labels and predictions further away will be ignored.

5.2.2 Data Association

To effectively compare labels and their corresponding predictions, they have to be matched together. Note, normally *IoU* values are calculated for this, however, due to the amplified ground-truth and prediction errors on the ground plane, a label and a prediction can still correspond to each other even if their *IoU* value is zero. Also, a prediction directly adjacent, but without any intersection with the label, has the same *IoU* value as a prediction far away. Therefore, the matching is done in the image and works as follows:

1. Calculate the 2D centers of the 3D bounding boxes by averaging the eight corners in the image plane
2. Compute pair-wise distances between of these centers for predictions and labels
3. Match a label and a predication together if they are both nearest to each other
4. Store the unmatched predictions and labels
5. Enforce the cutoff distance for the pairs and unmatched items

The matching is done before enforcing the cutoff because it may happen that e.g. a label is just outside the cutoff distance, while the corresponding prediction is just inside of it. If labels were removed before matching, the prediction would be counted as a false positive. The opposite direction would also lead to a wrong false negative, with switched locations of label and prediction.

5.2.3 Metrics

For each frame, the number of matches can be counted as true positives, the number of unmatched predictions as false positives, and the number of unmatched labels as false negatives. These can be summed up respectively and combined to micro-averaged *Precision*, *Recall*, and *F1-Score* values, as described in 2.4.

To compare the location and dimensions for each matched pair, *MAE* values are computed for x-location, y-location, length, width, and height. *MAE* values are chosen because they convey the intuitive interpretation: "How wrong (in meters) are my predictions are on average?", as described in Subsection 2.4.3.

Note, no *mAP* values are calculated here because of the mentioned issue with *IoU* values being zero.

5.2.4 Hyperparameter Optimization

Dimension Values Mapping

The first parameter is the mapping of categories to a minimum, maximum, and mean values for each dimension (i.e. length, width, and height). For that, all labels of the test set are used. Note, the minimum and maximum values are calculated as 2nd and 98th percentile to ignore outliers. Trailers and trucks are grouped together into a single mapping since in the *COCO* dataset these two are not differentiated. The heights are modified, as explained in Section 4.5. The mapping is equal for all instance segmentation models and can be seen in Table 5.2.

Category	Length [m]			Width [m]			Height [m]		
	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
BUS	10.29	13.27	17.60	2.50	2.60	2.81	2.83	3.33	3.93
CAR	3.33	4.60	4.88	1.57	1.95	2.17	1.22	1.45	1.82
TRUCK	2.57	9.30	16.72	2.01	2.52	2.92	2.83	3.60	4.21
VAN	4.10	5.80	7.45	1.67	2.06	2.51	1.82	2.18	2.83

Table 5.2: Mapping of vehicle categories to length, width, and height values calculated on a subset of the *A9-Dataset*. The minimum and maximum height values are aligned, such that cars, vans, and trucks/busses do not overlap.

Threshold Values

Additional hyperparameters are the three threshold values regarding the instance masks (i.e. score, width, and edge margin), as explained in 4.1. The first two are optimized using grid search with regards to the *F1-Score* based on matched pairs of predictions and labels within the cutoff distance. The edge margin is set to zero here because it is intended to remove

masks that are potentially cut off at the edges of the image, however, it would just decrease the recall here. In Table 5.3, you can see the optimal threshold values. Note, the mask width threshold is zero for two models and generally does not make a big impact here. It was originally intended to work as a proxy for the distance cutoff but due to the matched pairs, it does not have a substantial impact here. Nonetheless, in productive use, it can be really helpful.

Vertical Contour Shift

The last hyperparameter is the vertical contour shift of the bottom contour, as described in 4.2. It is optimized, with the previously determined threshold values, on the sum of location and dimensions *MAE* with the *L2*-distance as the absolute error term. In Table 5.3, it is visible that the shift is slightly more important for *YOLACT* models. This is because their produced masks are less accurate in general.

Model	Score	Mask Width [px]	Contour Shift [px]
Mask R-CNN R-50-FPN	0.62	20	2
Mask R-CNN X-101-FPN	0.61	25	2
Mask R-CNN X-152	0.52	20	2
YOLACT R-50-FPN	0.51	20	3
YOLACT R-101-FPN	0.51	0	3
YolactEdge R-50-FPN	0.52	20	3
YolactEdge R-101-FPN	0.52	0	3

Table 5.3: Optimal hyperparameters for the proposed approach per instance segmentation model.

5.2.5 Test Set Evaluation

With the optimal hyperparameters for each instance segmentation model, the produced predictions of the fixed angle approach can be evaluated on the test set. For this, recall, precision, and f1-score are calculated with the edge margin set to zero. The *MAE* values for location and dimensions are calculated with an edge margin of 10 pixels, which compensates for the inaccuracies produced by truncated vehicles at the image edges. This is needed because there are labels in the dataset with vertices outside of the image which would produce unfair false negatives. In Table 5.4, the results for each model are shown. The x-direction points along the road. All models perform very similarly, which is reasonable because only a 2D bounding box based on the bottom contour of the mask is used and therefore the lower mask accuracy of the *YolactEdge* models suffices.

In-depth Model Analysis

The *YolactEdge R-101-FPN* model performs slightly better than the *R-50-FPN* variant and is therefore the chosen model for real-time use. In the following, an in-depth analysis of the results by the *YolactEdge R-101-FPN* model on the test set is presented.

As mentioned in 5.1.3, only cars, trucks, and buses can be predicted as categories. However, based on the height estimation, as described in Section 4.5, the *VAN* category is also produced. In the confusion matrix for categories from labels to predictions, shown in Figure 5.4, it is clearly visible that this estimation generally works. However, the distinction between vans, trucks, and busses is still improvable.

Model	F1 ↑	Precision ↑	Recall ↑	*MAE* [m]↓ x	y	length	width	height
Mask R-CNN R-50-FPN	0.92	0.94	0.91	1.43	0.19	1.12	0.24	0.11
Mask R-CNN X-101-FPN	0.93	0.94	0.92	1.33	0.19	1.12	0.25	0.10
Mask R-CNN X-152	0.93	0.94	0.92	1.29	0.20	1.11	0.26	0.10
YOLACT R-50-FPN	0.91	0.93	0.88	1.57	0.22	1.08	0.24	0.12
YOLACT R-101-FPN	0.92	0.96	0.88	1.61	0.22	1.11	0.25	0.12
YolactEdge R-50-FPN	0.91	0.94	0.88	1.67	0.22	1.08	0.24	0.12
YolactEdge R-101-FPN	0.92	0.96	0.88	1.62	0.22	1.10	0.25	0.12

Table 5.4: Comparison of *MAE* values of the predictions on the test set using the fixed orientation approach per instance segmentation model.

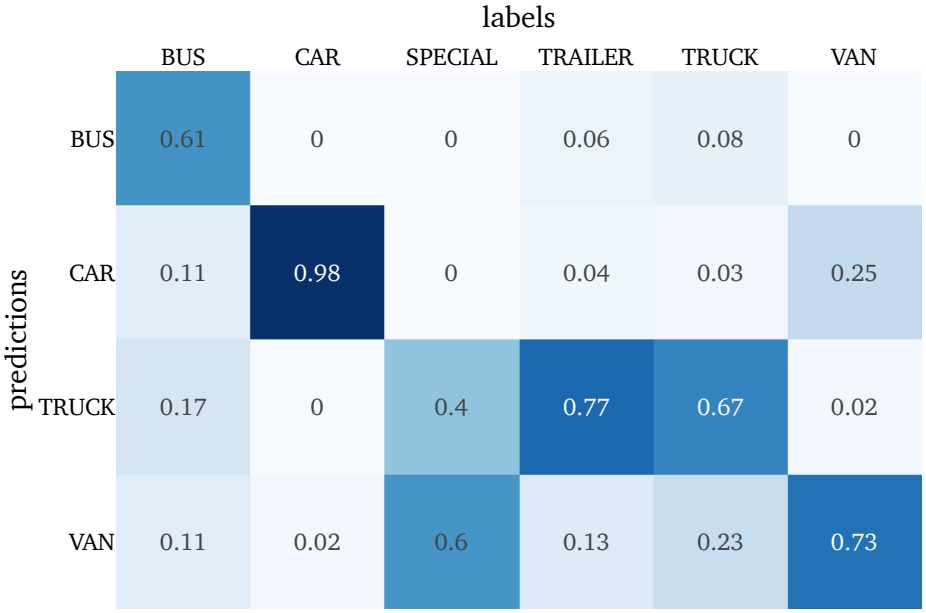


Figure 5.4: Confusion matrix for categories from labels (columns) to predictions (rows).

The *MAE* values can also be grouped by labeled category, as shown in 5.5. The number of matched vehicles for each category is also shown. Due to the larger dimensions, busses, trucks, and trailers have larger absolute errors for both location and dimensions. The very large x-location error is the result of a single outlier and the small number of busses. Also, the larger errors for truck and trailer length are greatly influenced by the labeling inconsistencies for these two categories, as mentioned in Subsection 5.2.1. The height estimation works reliably well for all classes.

The error values can also be grouped by the camera, which is shown in 5.6. Here, you can see that the cameras with a wider field of view (16 mm) have a larger error for length and a smaller x-location error. The reasonable assumption, that the accuracy decreases for a larger distance to the camera is confirmed, as shown in Figure 5.7. Here, the *MAE* values are also grouped per distance in 25-meter regions (i.e *MAE* value at 62.5 corresponds to the average from 50 to 75 meters). The errors per distance are generally higher for the 16 mm cameras because vehicles at the same distance are smaller pixel-wise compared to the 50 mm cameras. The latter also only captures vehicles that are further away than 50 meters.

		MAE [m] ↓				
Category	Count	x	y	length	width	height
BUS	18	16.83	0.51	3.79	**0.21**	0.41
CAR	2592	**1.22**	**0.19**	**0.48**	0.22	**0.10**
SPECIAL	5	2.56	0.32	5.15	0.37	**0.10**
TRAILER	170	2.99	0.43	4.53	0.38	0.23
TRUCK	117	4.75	0.42	7.82	0.35	0.36
VAN	347	2.14	0.29	1.65	0.35	0.14

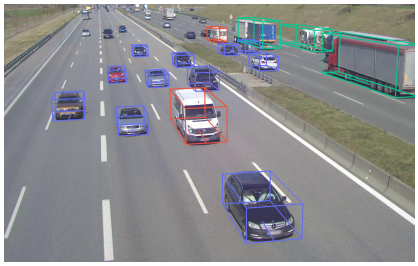
Table 5.5: *MAE* values and number of matched vehicles grouped by labeled category.

		MAE [m] ↓				
Camera	Count	x	y	length	width	height
S40 North 16 mm	752	1.46	0.27	1.31	0.24	0.16
S40 North 50 mm	981	1.59	0.21	**0.90**	0.27	0.11
S50 South 16 mm	391	**1.36**	**0.20**	1.32	0.30	0.15
S50 South 50 mm	1125	1.85	**0.20**	1.07	**0.22**	**0.09**

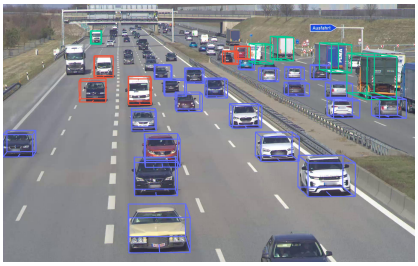
Table 5.6: *MAE* values and number of matched vehicles grouped by the camera.

5.2.6 Qualitative Results

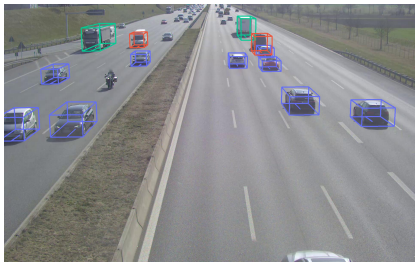
In Figure 5.5, you can see the results of the fixed angle approach with *YolactEdge R-101-FPN* from all four perspectives. In this case, a mask width threshold of 50 pixels is used. Cars are blue, vans are red, and trucks are green. A video is also available (https://youtu.be/I9XxiKNIwYE).



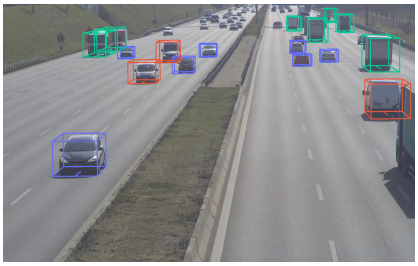
(a) S40 North 16 mm



(b) S40 North 50 mm

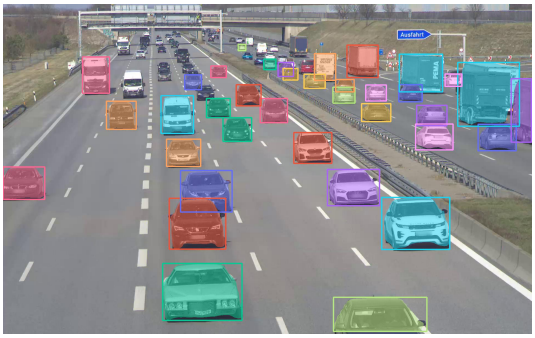


(c) S50 South 16 mm

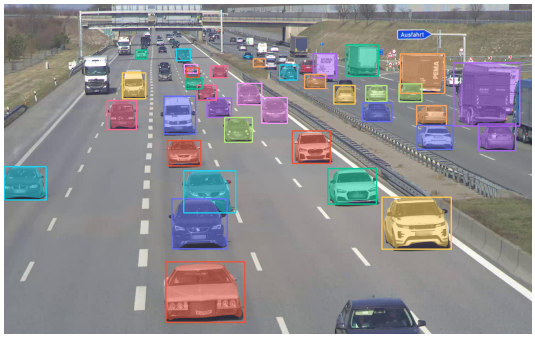


(d) S50 South 50 mm

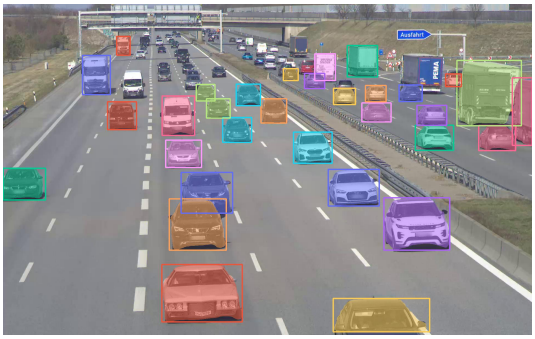
Figure 5.5: Qualitative results of the fixed orientation approach with *YolactEdge R-101-FPN* from all four perspectives of the *A9-Dataset*. Cars are blue, vans are red, and trucks are green.



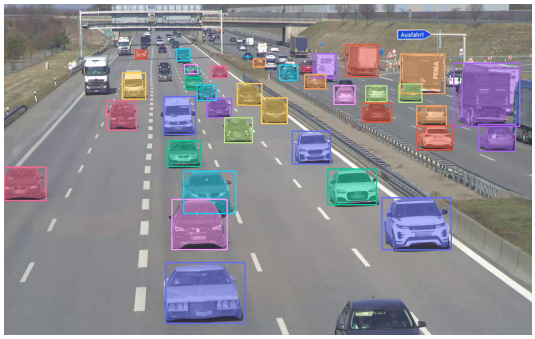
(a) *Mask R-CNN R-50-FPN*



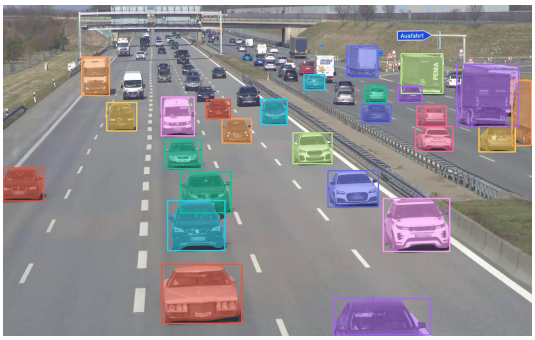
(b) *YOLACT R-50-FPN*



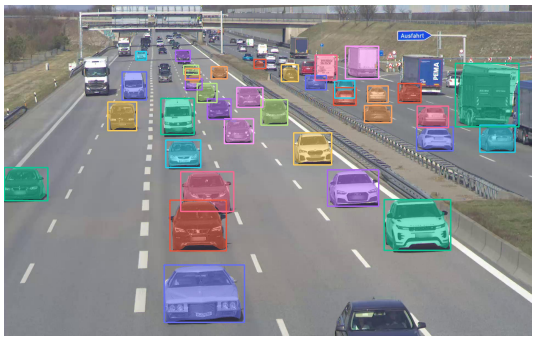
(c) *Mask R-CNN X-101-FPN*



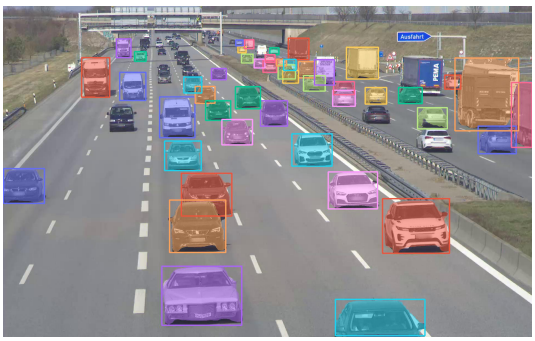
(d) *YOLACT R-50-FPN*



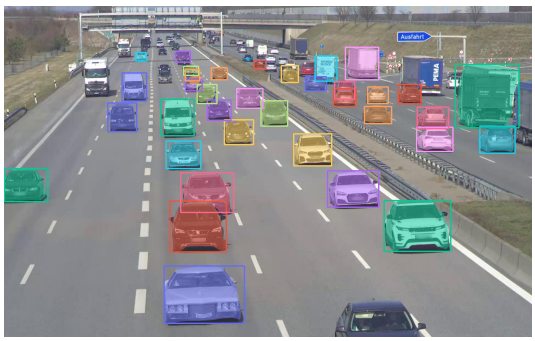
(e) *Mask R-CNN X-152*



(f) *YolactEdge R-101-FPN*



(g) *Mask R-CNN R-50-FPN**



(h) *YolactEdge R-101-FPN*

Figure 5.6: Qualitative instance segmentation results of pre-trained (*COCO*, **Cityscapes*) *Mask R-CNN* and *YOLACT* models above a 0.7 score.

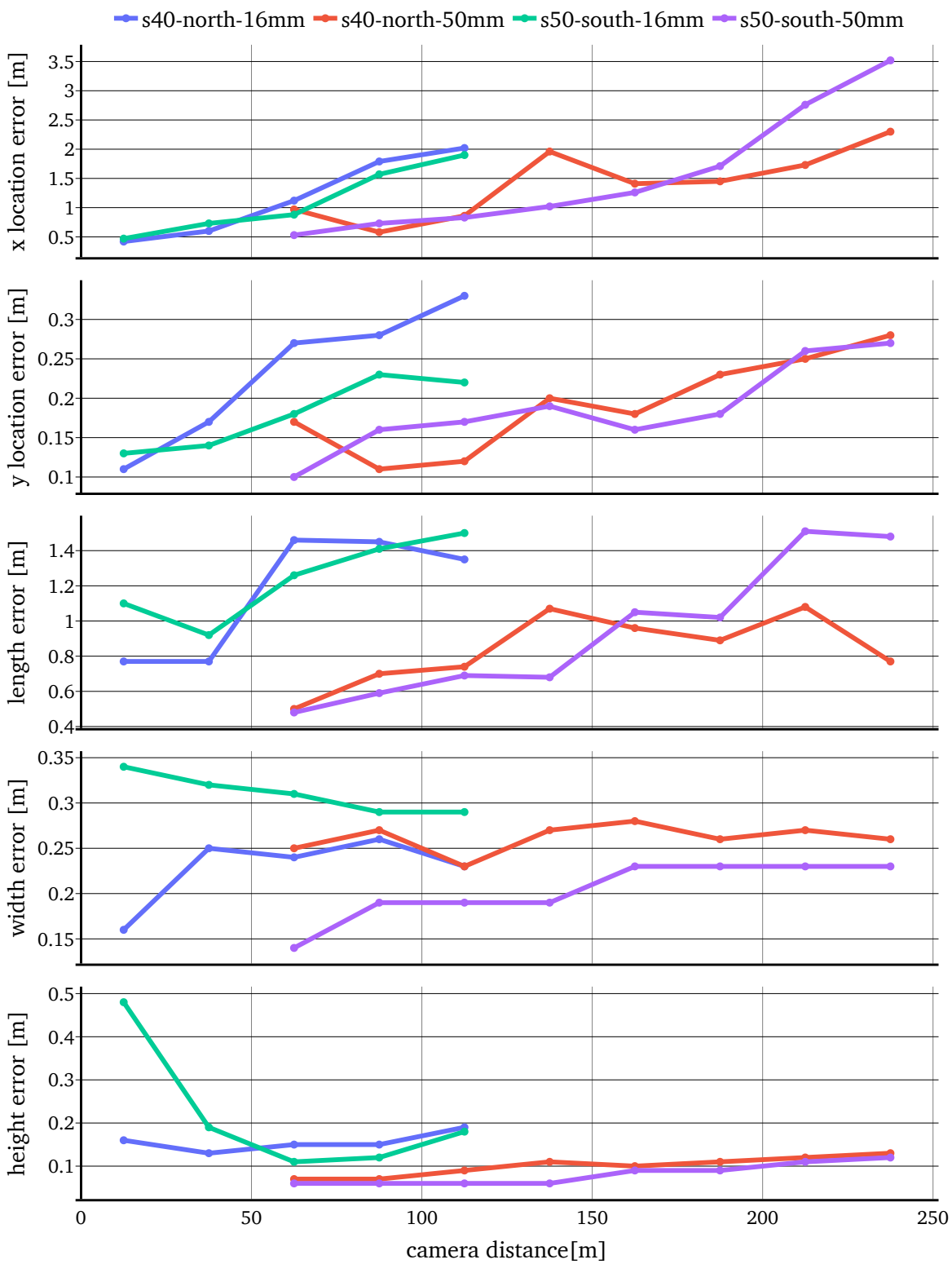


Figure 5.7: *MAE* values for location, length, width, and height grouped by distance to the camera of matched vehicles in the test set.

Chapter 6

Future Work

6.1 Dynamic Orientation

While the approach with a fixed orientation produces sufficient results for images of highways in the *A9-Dataset*, where the vehicle orientation is known, it cannot handle vehicles with dynamic orientations at an e.g. intersection. In Figure 6.1, a frame of an intersection is shown including the bottom contour points, which were lightly pre-processed by removing outliers at both ends and also interpolated.

Figure 6.1: A frame of an intersection at the *S110* measurement point of the extended *Providentia++* test stretch.

In comparison to the contours on the highway, as shown in Figure 4.5, here, the L-shape can be clearly seen. A 2D bounding box, aligned with the L-shape, was estimated with a minimization algorithm, which is shown in Figure 6.2. This gives each vehicle its own orientation. Similar to the fixed orientation approach, the corner nearest to the camera

should be the most accurate. This allows length and width estimation, such as in Section 4.3, for each vehicle, based on its specific orientation. The height estimation can also be used, as described in Section 4.4, though, the ratio between vehicle length and width has to be adjusted based on the orientation for each vehicle.

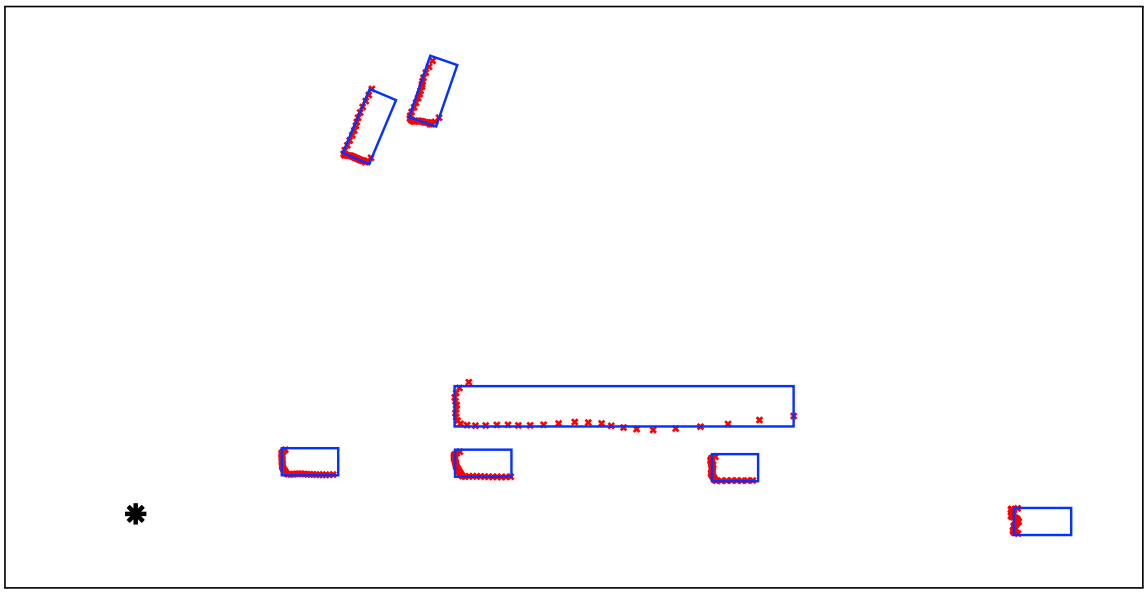


Figure 6.2: The ground contours and a 2D bounding box estimation for vehicles on a frame of an intersection at the *S110* measurement point of the extended *Providentia++* test stretch.

6.2 Combining Multiple Perspectives

With the bottom contour of a mask, the goal is to approximate the two nearest edges of the vehicle on the ground. Though, as shown in Figure 4.5, errors in the mask are amplified due to the projection onto the ground and lead to enlarged dimensions. However, the corner nearest to the camera is reasonably accurate. That brings up the idea to combine two or more cameras from different perspectives. For two directly opposing cameras, the nearest corners would also be on opposing sides, i.e span the diagonal line of the 2D bounding box on the ground. This is enough to accurately estimate the location and dimensions of a vehicle. However, both cameras need to be calibrated to fit perfectly together, a small deviation would make this approach useless.

6.3 Transfer Learning

As discussed in Subsection 5.1.3, the used instance segmentation models, pre-trained on the *COCO* dataset, do not cover all of the label categories in the *A9-Dataset*. A solution for that would be to fine-tune a model on all labeled images, which could lead to better classification results.

To improve the instance mask quality of *YolactEdge* models on the *A9-Dataset*, accurate instance masks could be created with a larger *Mask R-CNN* model, and then used to fine-tune the *YolactEdge* model. This could lead to better mask quality, tailored for vehicles on highways.

Chapter 7

Conclusion

In this thesis, an approach for monocular 3D object detection for vehicles on the *Providentia++* test stretch was proposed. It is also implemented in real-time and integrated it into the existing *Providentia++* toolchain using *ROS*.

First, it was presented that the majority of existing work in the domain of monocular 3D object detection is created for *AVs* from the ego perspective. However, these approaches generally cannot translate their success to cameras from the road-side perspective. Furthermore, it would be unfavorable to not leverage the fact, that the cameras on the test stretch are precisely calibrated, fixed in their position, and only capture vehicles on a straight highway segment.

Therefore, a method that takes advantage of the road-side perspective and uses pre-trained instance segmentation models as a backbone is proposed. These reliably detect the majority of the visible vehicles on an image and create instance masks and class labels for each of them. Next, the bottom contours of these masks are extracted, which should approximate two of the lower edges on the ground of the vehicles. These are then projected onto the ground plane using the calibrated data of the camera. Using the fact that the orientation of the vehicles is fixed, due to the straight highway, the projected contours can be used to estimate the location, length, and width of the vehicles. Additionally, the heights of the vehicles are estimated based on the height of the instance mask and the viewing angle of the camera, using a geometric argument. This last step also allows the classification of vehicles as vans, which the instance segmentation models, pre-trained on the *COCO* dataset, are not capable of. This works because the height of vans is right between cars and trucks/busses.

To show that the proposed method is working, it was evaluated on images of the *A9-Dataset*. This was done with two different instance segmentation models, namely *Mask R-CNN* and *YolactEdge*. However, only the *YolactEdge* variants, using *TensorRT*, are capable of running in real-time. When compared to the *Mask R-CNN* models on instance segmentation on the *COCO* dataset, they perform notably worse, however, when used as the backbone for predicting the 3D bounding boxes in the *A9-Dataset*, their performance is very similar. In general, the predicted 3D bounding boxes are sufficiently close to their corresponding labels, especially the height estimation works reliably well. Also, 70 % of the labeled vans are actually classified as vans.

Nonetheless, the approach is not flawless. The performance of the instance segmentation models is greatly influenced by the image quality. Bad weather and unfavorable lighting conditions decrease the performance substantially. Also, occluded or truncated vehicles result in incomplete masks, which are not handled by the proposed method. Additionally, it is only applicable to the highway scenarios, though, as part of the future work, it was explained how a single change, i.e. the dynamic estimation for the vehicle orientations, can improve the method, such that it can also be used from other perspectives, e.g. an intersection.

Bibliography

[Cae+20] Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. "nuScenes: A multimodal dataset for autonomous driving". In: (May 2020). URL: http://arxiv.org/abs/1903.11027.

[Che+21] Cheng, B., Misra, I., Schwing, A. G., Kirillov, A., and Girdhar, R. "Masked-attention Mask Transformer for Universal Image Segmentation". In: *arXiv* (2021).

[CBL19] Clausse, A., Benslimane, S., and La Fortelle, A. de. "Large-Scale extraction of accurate vehicle trajectories for driving behavior learning". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. ISSN: 2642-7214. June 2019, pp. 2391–2396. DOI: 10.1109/IVS.2019.8814095.

[Cor+16] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. "The Cityscapes Dataset for Semantic Urban Scene Understanding". en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 3213–3223. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.350. URL: http://ieeexplore.ieee.org/document/7780719/.

[Cre+22] Creß, C., Zimmer, W., Strand, L., Fortkord, M., Dai, S., Lakshminarasimhan, V., and Knoll, A. "A9-Dataset: Multi-Sensor Infrastructure-Based Dataset for Mobility Research". en. In: (2022), p. 6.

[GLU12] Geiger, A., Lenz, P., and Urtasun, R. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[Gir15] Girshick, R. "Fast R-CNN". en. In: (Apr. 2015). URL: https://arxiv.org/abs/1504.08083v2 (visited on 02/11/2022).

[Guo+21] Guo, E., Chen, Z., Rahardja, S., and Yang, J. "3D Detection and Pose Estimation of Vehicle in Cooperative Vehicle Infrastructure System". In: *IEEE Sensors Journal* 21.19 (Oct. 2021). Conference Name: IEEE Sensors Journal, pp. 21759–21771. ISSN: 1558-1748. DOI: 10.1109/JSEN.2021.3101497.

[HZ04] Hartley, R. and Zisserman, A. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press, 2004. DOI: 10.1017/CBO9780511811685.

[He+18] He, K., Gkioxari, G., Dollár, P., and Girshick, R. "Mask R-CNN". In: (Jan. 2018). URL: http://arxiv.org/abs/1703.06870.

[Kra+] Krammer, A., Scholler, C., Gulati, D., and Knoll, A. "Providentia - A Large Scale Sensing System for the Assistance of Autonomous Vehicles". en. In: (), p. 5.

[Li+20] Li, P., Zhao, H., Liu, P., and Cao, F. "RTM3D: Real-time Monocular 3D Detection from Object Keypoints for Autonomous Driving". In: (Jan. 2020). URL: http://arxiv.org/abs/2001.03343.

[Lin+15] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. “Microsoft COCO: Common Objects in Context”. In: (Feb. 2015). URL: http://arxiv.org/abs/1405.0312.

[Liu+21a] Liu, H., Soto, R. A. R., Xiao, F., and Lee, Y. J. “YolactEdge: Real-time Instance Segmentation on the Edge”. In: *ICRA*. 2021.

[Liu+21b] Liu, Z., Zhou, D., Lu, F., Fang, J., and Zhang, L. “AutoShape: Real-Time Shape-Aware Monocular 3D Object Detection”. In: (Aug. 2021). URL: http://arxiv.org/abs/2108.11127.

[Par+21] Park, D., Ambrus, R., Guizilini, V., Li, J., and Gaidon, A. “Is Pseudo-Lidar needed for Monocular 3D Object detection?” In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021.

[Red+16] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. “You Only Look Once: Unified, Real-Time Object Detection”. In: *arXiv:1506.02640 [cs]* (May 2016). URL: http://arxiv.org/abs/1506.02640.

[RAM21] Rezaei, M., Azarmi, M., and Mir, F. M. P. “Traffic-Net: 3D Traffic Monitoring Using a Single Camera”. In: (Sept. 2021). URL: http://arxiv.org/abs/2109.09165.

[Sun+20] Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., et al. “Scalability in perception for autonomous driving: Waymo open dataset”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2446–2454.

[Wan+18] Wang, Y., Chao, W.-L., Garg, D., Hariharan, B., Campbell, M., and Weinberger, K. Q. “Pseudo-LiDAR from Visual Depth Estimation: Bridging the Gap in 3D Object Detection for Autonomous Driving”. en. In: (Dec. 2018). URL: https://arxiv.org/abs/1812.07179v6.

[Wu+19] Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., and Girshick, R. *Detectron2*. 2019. URL: https://github.com/facebookresearch/detectron2.

[Zho+] Zhou, Y., He, Y., Zhu, H., Wang, C., Li, H., and Jiang, Q. “Monocular 3D Object Detection: An Extrinsic Parameter Free Approach”. en. In: (), p. 11.

[Zhu+22] Zhu, M., Zhang, S., Zhong, Y., Lu, P., Peng, H., and Lenneman, J. “Monocular 3D Vehicle Detection Using Uncalibrated Traffic Cameras through Homography”. In: (Jan. 2022). arXiv: 2103.15293. (Visited on 02/12/2022).