Moritz Kampelmühler

Estimating the velocity of vehicles relative to a moving camera

Master's Thesis

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme: Information and Computer Engineering

submitted to
Graz University of Technology

Supervisors:
Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Axel Pinz
Dipl.-Ing. Dr.techn. Christoph Feichtenhofer

Institute for Electrical Measurement and Measurement Signal Processing

Faculty of Computer Science and Biomedical Engineering

Graz, April 2018

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date

Signature

Abstract

This thesis was originally motivated by the *CVPR'17 Velocity Estimation Challenge*, the goal of which was to estimate the relative velocity of vehicles with respect to a reference vehicle equipped with a monocular camera in freeway traffic. Our initial method, which consists of first extracting tracks, dense depth maps and dense optical flow, and then estimating velocities from these features using a neural network, has won the challenge. Yet, a lot of questions remained unanswered at the point of submission, most notably how useful the features employed in the initial method are for the task and if there are other features or even other methods that are more powerful.

This thesis aims at answering those questions by conducting a thorough ablation study over the different features and neural network architectures used and also adding an object detection stage for bounding box tracking and semantic segmentation as an additional feature. The ablation study investigates the velocity estimation performance for all individual input features as well as combinations of various features. Furthermore, I introduce two velocity estimation methods based on simple geometry that serve as a baseline to compare the learning based methods to.

My results demonstrate that tracking is the most useful feature for vehicle velocity estimation and that features generated by convolutional neural networks are only reliable for vehicles in close ranges of the reference vehicle. The learning based methods are performing generally well, however they are not able to outperform geometry based baseline methods for vehicles at certain ranges. Using a tracking based approach, a computationally lightweight and efficient system for vehicle velocity estimation can be developed, which is crucial in autonomous driving scenarios.

Kurzfassung

Die ursprüngliche Motivation dieser Arbeit stammt von der *CVPR'17 Velocity Estimation Challenge*, deren Aufgabenstellung darin bestand, die relative Geschwindigkeit von Fahrzeugen auf Autobahnen aus der Sicht eines Referenzfahrzeuges, welches mit einer einzelnen Kamera ausgestattet ist, zu schätzen. Mit unserer Methode konnten wir die Challenge gewinnen. Diese Methode bestand daraus zuerst für die Geschwindigkeitsschätzung relevante Parameter, nämlich Tracks, monokulare Disparitäten und optischen Fluss, zu extrahieren und dann Geschwindigkeiten unter Verwendung von künstlichen neuronalen Netzen zu schätzen. Zum Zeitpunkt der Einreichfrist für die Challenge blieben jedoch einige Fragen unbeantwortet, allem voran wie sinnvoll die genutzten Parameter für die Aufgabe sind und ob bessere Parameter oder gar Methoden entwickelt werden können.

Diese Arbeit versucht diese Fragen zu beantworten, indem eine umfassende Studie durchgeführt wird, die den Einfluss der einzelnen Parameter und unterschiedlicher Architekturen auf die Qualität der Geschwindigkeitsschätzungen untersucht. Weiters wird ein Objekterkennungsalgorithmus hinzugefügt mit dem sowohl Tracks als auch Segmentierungsmasken erzeugt werden können. In der Studie werden sowohl alle Parameter einzeln als auch unterschiedliche Kombinationen betrachtet und deren Nützlichkeit evaluiert. Um die auf neuronalen Netzwerken basierenden Methoden besser vergleichen zu können werden auch geometriebasierte Methoden zur Geschwindigkeitsschätzung entwickelt.

Die Ergebnisse zeigen, dass Tracks eindeutig den höchsten Informationsgehalt für die Geschwindigkeitsschätzung haben und dass Methoden die auf Convolutional Neural Networks basieren nur für Schätzungen geeignet sind, wenn die betreffenden Fahrzeuge nahe an der Kamera sind. Neuronale Netzwerke erzielen generell sehr zufriedenstellende Ergebnisse, wobei sie nicht in der Lage sind für Fahrzeuge in mittleren Distanzen genauere Schätzungen zu erzielen als geometriebasierte Methoden. Unter der Verwendung von Methoden, die auf Tracks basieren, können sehr effiziente Systeme für die Geschwindigkeitsschätzung mit geringem Rechenaufwand entwickelt werden, was für autonomes Fahren eine entscheidende Rolle spielt.

Acknowledgments

First and foremost, I want to thank my mother, father, sister and Michael for their perduring love, support and encouragement.

Thank you Axel and Christoph for the constructive criticism, help, advice and for motivating me to get into research.

Thank you Marian for being the person you are and being a part of my life.

Thank you Michi for having my back at all times and for being unfazed by my unequaled rigor (or lack thereof). You are one of the reasons I made it to where I am today.

Thank you Michi for simply being a great bloke and a faithful soul.

Thank you Vale for always being there when I need you.

Thank you Martina for constantly pushing me. If it weren't for you, I would probably still be stuck in my undergrad.

Thank you Clemens for always being honest, joyful, empathetic, and of course the main man.

During the research and writing of this thesis, I have lived through one of the most difficult periods in my life. I want to express my heartfelt gratitude to all the people who have stood by me: Elfi, Karl, Thomas, Katrin, Simon, Michele, Petra, Paco, Felix, Fabian, Manuel, Dominik and all the others not explicitly mentioned here.

Dedicated to my brother Paul, Rest in Peace.

Contents

1 Introduction

Autonomous driving is without a doubt one of the hottest topics in automotive research, which poses great challenges in many different fields. It requires low-cost, low-power and real-time capable solutions for joint sensing and control. Current vehicles used for autonomous driving benchmarks are often equipped with a multitude of sensors, like for example two color and two grayscale video cameras, a laser scanner and GPS/IMU localization units (Geiger et al., 2012). In this work, the focus lies on vision based sensing using only a single camera.

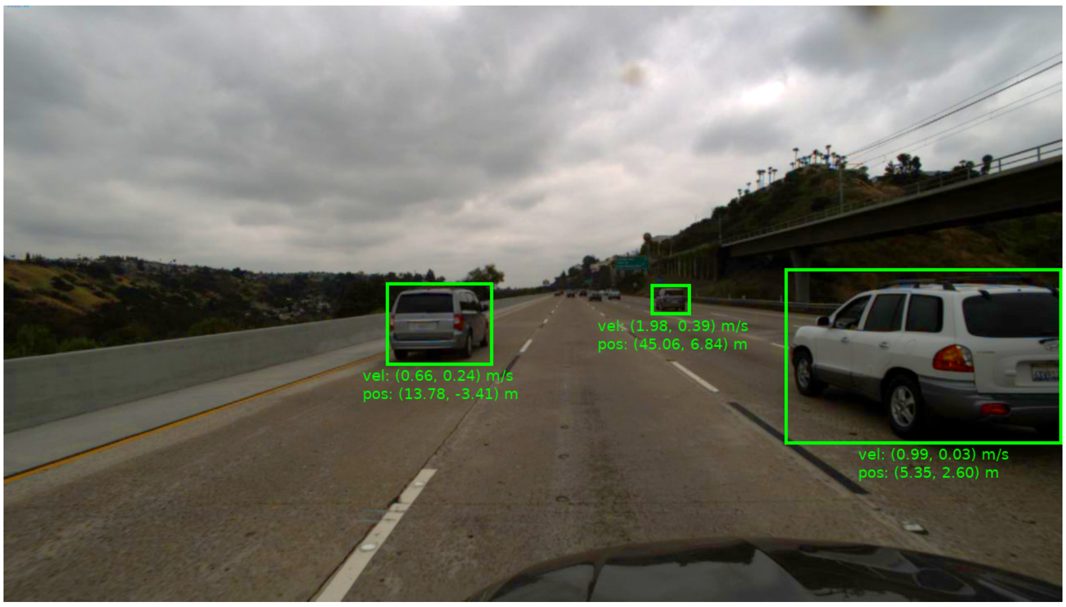

Figure 1.1: A sample image with annotations from a training sequence. Velocity and position ground truth are provided for the vehicles surrounded by the green bounding boxes in both longitudinal (X) and lateral (Y) directions (see Figure 2.1).

1 Introduction

Camera sensors provide an inexpensive yet powerful alternative to range sensors based on LiDAR or radar. While LiDAR systems can provide very accurate measurements, they may also malfunction under adverse environmental conditions such as fog, snow, rain or even exhaust gas fumes (Rasshofer et al., 2011; Hasirlioglu et al., 2017). Arguably, vision based sensing is more closely related to how humans engage in driving situations and it should thus be possible to solve any task in autonomous driving based on visual input.

This work addresses monocular vehicle velocity estimation, an emerging task in autonomous driving which has not yet been thoroughly explored. The specific task, which forms the base for this work, was introduced as the Velocity Estimation Challenge[1] at the CVPR2017 Autonomous Driving Workshop. The goal is to estimate the relative velocity of a specific vehicle from a sequence of monocular RGB images to aid autonomous driving algorithms such as for example collision avoidance (Aufrère et al., 2003) or adaptive cruise control (Jurgen, 2006). Figure 1.1 shows an example image from the data.

Vehicle velocity estimation as such is not a new subject of interest, since it is extensively studied in the context of traffic surveillance (Hsieh et al., 2006; Coifman et al., 1998), where, however, a stationary camera is employed. Under the restriction of a fixed camera pose, the problem becomes significantly less complex, since with a calibrated camera system angular measurements can be obtained and from these measurements velocity estimates can readily be established. In contrast, in our case the observer resides on a moving platform and inferring velocity in a similar fashion would require additional information such as camera pose, ego-motion and foreground-background segmentation. Very recent research (Zhou et al., 2017) shows that estimating ego-motion as well as disparity maps from monocular camera images by means of *structure from motion* is indeed possible, but still limited. Semantic segmentation of scenes, which is a fundamental problem in computer vision, has also more recently been tackled using deep neural networks (Dai et al., 2016; Li et al., 2016).

In a more general sense, the given task can be seen as a lightweight version of *object scene flow* as for example provided in the KITTI benchmark (Geiger et al., 2012; Menze and Geiger, 2015). Object scene flow aims at estimating dense 3D motion fields, which in their temporal evolution carry highly valuable information about the geometric constellation of a given scene. Recent approaches (Vogel, Schindler,

[1] `http://benchmark.tusimple.ai/#/t/2`

et al., 2013; Vogel, Roth, et al., 2014) yield impressive results, but they rely on the availability of stereo image data. Furthermore, they come at the price of very high computational cost, such that the estimation for a temporal frame pair might take 5-10 minutes on a single CPU core. In autonomous driving scenarios, computational resources are in general highly limited (Grzywaczewski, 2017), which makes object scene flow currently not practically feasible.

In this work, we adopt recent deep learning architectures (Ilg et al., 2017; Godard et al., 2017) for depth and motion estimation to leverage a mapping of the video input into a beneficial feature space for learning from the few training samples provided. Our approach employs a two-stage process for monocular velocity estimation. In a first step we extract vehicle tracks as well as dense depth and optical flow information, followed by locally aggregating these depth and motion cues at the tracked vehicle locations and concatenating over the temporal dimension. After this feature extraction procedure we use the spatiotemporal depth, flow and location features to train a fully connected regression network for velocity estimation of the respective vehicles.

Further on, we conduct an extensive ablation study to investigate the impact of the individual features and combinations thereof on the regression performance as well as on the runtime of the estimation. We show that a light weight implementation can achieve excellent results, and that leveraging deep motion and depth cues does not necessarily improve performance for this task on the given data.

Problem Statement

The problem tackled in this thesis was originally posed as the *CVPR'17 Velocity Estimation Challenge*. Vehicle velocity estimation aims at inferring the relative velocity of cars as seen from a reference car equipped with a single camera. For the estimation short video snippets with annotated velocity and position ground truth are available (see Chapter 2 for more details). Figure 1.1 shows an example from the dataset, with three annotated vehicles at different ranges and their ground truth velocities and positions. Note that for each video sequence a single velocity value is annotated per car and a single value has to be inferred, *i.e.* not every frame and every vehicle is annotated, but a single velocity value is representative for the whole sequence. For inference, the initial location of the car of which the velocity should be estimated is given, thus no detection is necessary.

1 Introduction

Within my work on this Master's Thesis, two scientific contributions have been achieved:

- Winner of the *CVPR'17 Velocity Estimation Challenge*.
- Conference paper (Kampelmühler et al., 2018) that has won the best student paper award at CVWW'18.

2 The *CVPR'17 Velocity Estimation Challenge* Dataset

The provided training dataset includes 1074 short driving sequences in freeway traffic, recorded by a single HD (1280x720p) camera. Each of those sequences spans 40 frames over a 2 second time frame, resulting in a 20 frames per second framerate. Additionally the intrinsic camera calibration parameters (K matrix) as well as the height H above ground of the camera, are provided as supplementary information. For each sequence, only specific vehicles are annotated.

An annotation is provided for the last frame of each sequence only and consists of a bounding box (pixel coordinates) as well as position (in m) and velocity (in m/s) in both X and Y coordinates, where Y refers to the point of the vehicle closest to the camera. Spatially, the vehicle coordinate system is oriented as schematically shown in Figure 2.1; X is in line with the optical axis and Y is perpendicular to X, pointing towards the vehicle's right. The challenge organizers have acquired ground truth velocity and position data using a fusion of LiDAR, radar and stereo vision. They have communicated a ground truth accuracy of 0.5 $\frac{\mathrm{m}^2}{\mathrm{s}^2}$ velocity MSE (see Section 5.2 for details).

A bounding box annotation is a list consisting of four characteristic coordinates `[left, right, top, bottom]` specifying the border coordinates of the bounding box in pixels. An example annotation for the vehicle framed by a light green bounding box in Figure 2.1 is given in Table 2.1. This annotation tells us that the vehicle is 22 m in front of our car, 6 m off to our right side and moving at approximately 2 m/s slower than our car in approximately the same direction, slowly closing in from the right side.

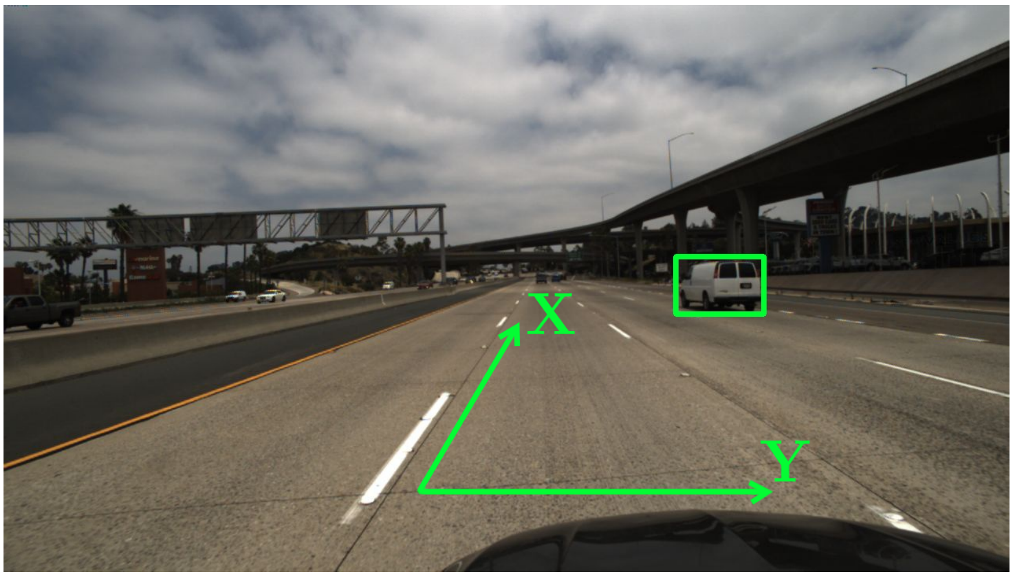

Figure 2.1: A sample from the dataset with coordinate axes X and Y indicated

velocity		position	
X	Y	X	Y
-2.07 m/s	-0.26 m/s	22.14 m	5.97 m
bounding box			
left	right	top	bottom
859 px	965 px	323 px	389 px

Table 2.1: Annotation example for Figure 2.1

For each training sequence, up to 4 vehicles are annotated which in total gives 1442, vehicle annotations to work with, an average of 1.34 per sequence. For evaluation the individual vehicles are classified into three clusters according to their ground truth relative distance d_{gt} in the last frame. $d < 20$ m is considered near range ($\sim$ 12% of samples), 20 m $\geq d >$ 45 m medium range ($\sim$ 65% of samples) and $d \geq$ 45 m far range ($\sim$ 23% of samples). Figure 2.2 shows the distribution of annotated vehicles with respect to their relative distance. Clearly, most vehicles belong to medium range, which makes sense considering that a 2 s safety distance (approx. 44 m at 80 km/h) is mandatory.

For each of the aforementioned ranges, different difficulties arise in the estimation. In near range examples the perspective on vehicles can shift drastically in between instances and over time for individual samples. For far range samples the pixel resolution of the data limits the estimation capabilities. Both of the ‘extreme’ ranges come with the additional difficulty of few training samples.

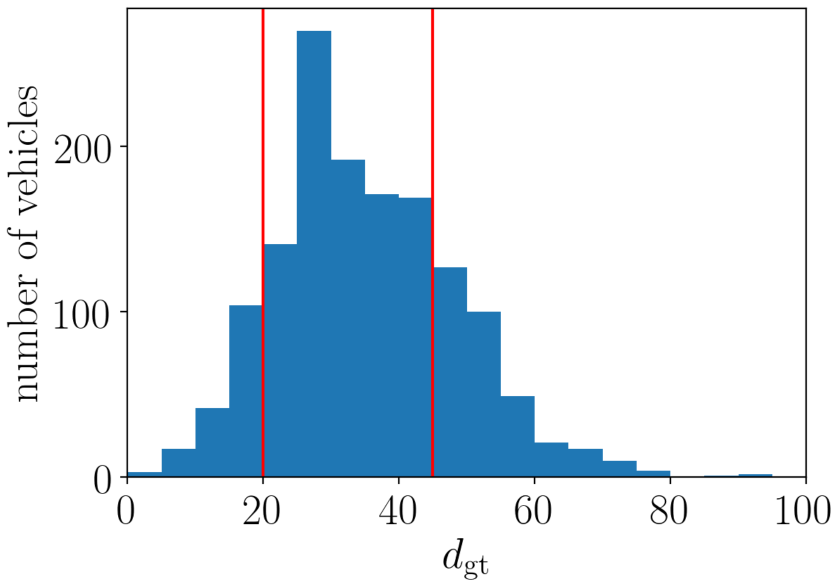

Figure 2.2: Histogram of ground truth distances of annotated vehicles in last frame. The vertical lines indicate the borders between near, medium and far ranges.

In addition to the training data, a supplementary set is available, which can be used to train a car detector. The supplementary data includes 5067 frames, for which 29115 bounding boxes are annotated. Since I do not train a car detector for my approach, I have not used this data.

The evaluation for the challenge leaderboard is carried out on a separate test set, with unknown position and velocity annotations. Here, the organizers of the challenge provide only a bounding box, again for the last frame. This set comprises 269 clips with 375 annotated vehicles.

3 Related work

For my solution to the problem posed as the *CVPR'17 Velocity Estimation Challenge*, I employ a variety of different methods that emerged from past computer vision research. Hence, this section gives an overview of related work on the methods used, namely Tracking, Monocular Depth estimation, Optical flow estimation as well as Object Detection and Segmentation (Sections 3.1-3.4). Additionally, Section 3.5 presents alternative solutions to the *CVPR'17 Velocity Estimation Challenge* as submitted by other contestants.

3.1 Tracking

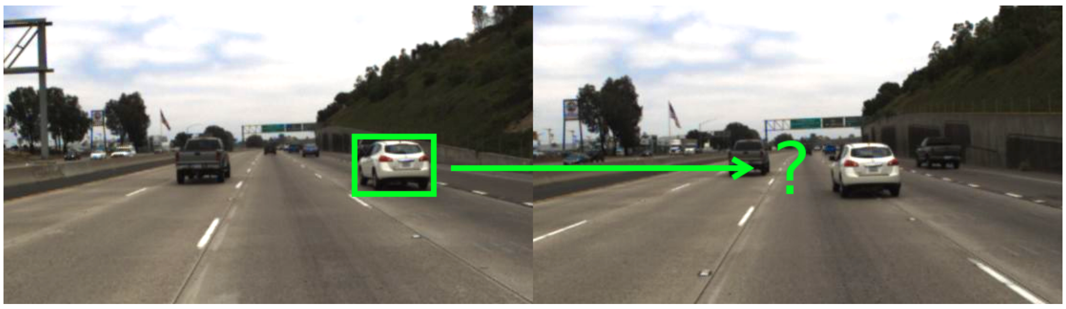

Figure 3.1: Basic concept of object tracking. The highlighted car on the left has to be found in the picture on the right.

Object tracking is one of the fundamental problems in computer vision and has been extensively studied (Yilmaz et al., 2006) and applied in many different tasks such as traffic surveillance. The aim of tracking is to preserve the identity of one or more objects or points of interest over time. Hence, the basic problem is formulated as follows: Given two frames, frame A and frame B, and a reference area in frame A, decide whether the object included in the reference area is also present in frame B and if so where it is located. An example is given in Figure 3.1, where the location of the vehicle denoted by a bounding box in the left frame is unknown in the right frame. Some of the main challenges in tracking tasks are posed by occlusions, large displacements, changes in appearance of non rigid objects and long term preservation of identity (Lukežič et al., 2017).

One widely used method that I also use in this thesis is *Median Flow* (Kalal et al., 2010). *Median Flow* is an approach building on top of the *Lucas-Kanade* (Lucas and Kanade, 1981) method, which is an early optical flow algorithm operating on local intensity changes. This method is extended by a Forward-Backward error, which denotes the deviation between the trajectories obtained by tracking from $I_{t-1} \rightarrow I_t$ and $I_t \rightarrow I_{t-1}$ (*i.e.* forward and backward through time). Good points to track are identified by evaluating the forward backward error for every tracked point and keeping the ones that can be tracked reliably. For bounding box tracking, this is done for points within the reference bounding box. Forward-Backward error is thus able to detect tracking failures caused by *e.g.* occlusions. Another common approach is the so called *Multiple Instance Learning* or short *MIL* tracker (Babenko et al., 2009). This method first transforms the image into an appropriate feature space, and uses a classifier as well as a motion model to determine the presence of an object in a frame, which falls into the category of 'tracking by detection' approaches.

More recent methods like (Held et al., 2016) employ convolutional neural networks to learn motion and appearance of objects. The feature maps of higher convolutional layers provide robust and accurate appearance representations, but lack spatial resolution. Lower layers on the other hand provide higher spatial resolution and less refined appearance representations. This hierarchical structure is used in (Ma et al., 2015) by inferring responses of correlation filters on each corresponding layer pair. Other neural network based approaches employ recurrent structures such as LSTMs to tackle long term tracking (Sadeghian et al., 2017).

3.2 Monocular Depth Estimation

Monocular Depth Estimation aims at inferring the depth of a scene, *i.e.* how far each pixel of an image is away from the camera. Traditionally, estimation of depth is handled by employing LiDAR sensors (laser scanners) or stereo camera rigs. LiDAR sensors are expensive to acquire and operate and high definition stereo video elicits a large flow of data. In autonomous driving scenarios, real time capability with very limited computational resources is crucial, which is why operation on monocular images, *i.e.* images taken with a single camera, is often used (*e.g.* Mobileye®[1]). Estimating the depth information of a scene seen from a given angle using only a single camera is not a well-posed problem. Most of the methods that are currently used to work on this problem are based on deep convolutional neural networks with large amounts of training data, since depth information can be inferred from single views by appearance which requires ample experience.

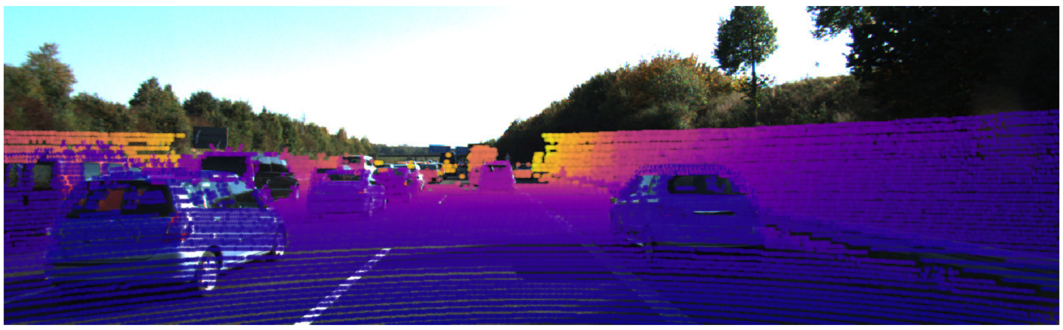

Figure 3.2: Sample image with LiDAR points projected into the image. A limited amount of high precision depth measurements is acquired per frame.

Some of the recent methods tackling this problem (Saxena et al., 2006; Xu et al., 2017) apply supervised learning regimes requiring ground truth depth data. The acquisition of such ground truth data is generally not a simple task either and is most often performed by a fusion of stereo vision and LiDAR point clouds. It requires the synchronization of cameras and rotating laser scanner, stereo inference, mapping of the sparse LiDAR point cloud into the stereo depth image and subsequent interpolation. An example of a sparse laser scanner point cloud projected into an image is given in Figure 3.2; the RGB image and LiDAR data are taken from the KITTI dataset (Geiger et al., 2012). While LiDAR sensors are very accurate ($\sim$2 cm position accuracy, Velodyne HDL-64E), they only provide sparse point clouds of around 120k points around the vehicle, out of which only a fraction overlaps with the frames recorded by the camera; *e.g.* in Figure 3.2 18033 LiDAR measurement points fall into the frame of $1241 \times 376 = 466616$ px. Due to the effort associated with obtaining high quality dense depth ground truth, its availability is very limited.

[1]`https://www.mobileye.com/`

Hence several methods are being developed that require little to no supervision.

(Kuznietsov et al., 2017) describe a semi-supervised approach that provides a fusion of using sparse ground truth data from a LiDAR sensor and stereo view synthesis. The idea behind stereo view synthesis is to estimate one image in a stereo pair from the other. Being able to estimate synthetic stereo images also enables one to infer stereo disparities and thus depth maps. Other similar approaches (Godard et al., 2017; J. Xie et al., 2016) in turn rely solely on stereo view synthesis with stereo image pairs as a supervision signal, which comes with the inherent benefit of easily available or obtainable data as well as the independence of LiDAR systems. Note that deep learning approaches in general benefit from large datasets with high variability.

Some recent work (Zhou et al., 2017; Garg et al., 2016) shows that deep learning frameworks are capable of inferring single image depth from monocular video as sole supervision signal. They achieve this by leveraging a learning enabled structure from motion type approach. The (mostly small) temporal motion of the camera and its thus changing pose provides multiple views of a given scene. Via novel view synthesis the future camera frames can be predicted and the actual future frames can be used as a supervisory signal. The mapping thus learned again implicitly carries information about the 3D scene geometry. Comparing to the approaches using stereo image pairs for training, this method exploits the temporal rather than the spatial dimension.

3.3 Optical Flow Estimation

Optical flow is another key research area in computer vision, *e.g.* for video object detection (Zhu et al., 2017), to quantify pixel-wise motion in between frames of a moving scene. In other words, optical flow can be referred to as dense pixel level tracking, with the key difference to object tracking being that it does not make considerations about the object appearance. The goal is to assign a motion vector (u,v) to each pixel to encode its displacement from one frame to the other. For the purpose of visualization, usually the HSV colorspace is used, where the hue represents the direction and the value represents the magnitude of the movement vector. This is called the ‘Middlebury encoding’ scheme (Baker et al., 2011). A result of such a visualization is shown in Figure 3.3, where the relation of the different angles to the hue can clearly be seen.

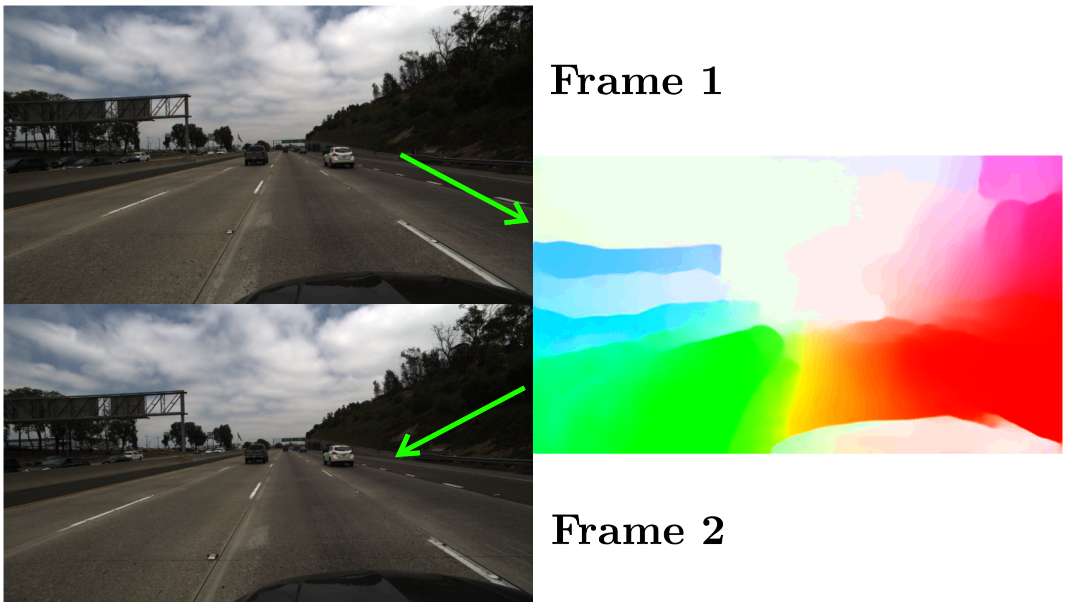

Figure 3.3: Two consecutive frames (left) with optical flow (right) in Middlebury (Baker et al., 2011) encoding.

Traditional methods (Horn and Schunck, 1981; Deriche et al., 1995) employ variational motion estimation approaches that regard the estimation of pixel level correspondences between frames as an optimization problem; *e.g.* Horn and Schunck, 1981 calculate optical flow components u and v locally from brightness gradients under the constraint of quadratic smoothness. The smoothing is required to counteract discontinuities in the dense optical flow field.

With the growing interest in deep learning, optical flow estimation is now most often also successfully treated as a supervised learning problem (Dosovitskiy, Fischer, Ilg, Hausser, et al., 2015). This is achieved by employing encoder-decoder structures that use a convolutional neural network for feature extraction and aggregation, followed by an ‘upconvolutional’ network. The ‘upconvolutional’ network concatenates the feature maps from the corresponding convolutional layers and jointly applies fractionally strided convolution to increase spatial resolution back to input size of the feature extraction network. Further improvements on this approach have since been made (Ilg et al., 2017) that provide improved performance, robustness as well as scalability. Those improvements consist of efficient dataset schedules for learning, stacking of multiple dedicated FlowNets (with and without correlation layers) and combining architectures for small and large displacements.

3.4 Object detection and segmentation

The goal of object detection in computer vision is to detect the presence and location of one or more diverse categories of objects (*e.g.* whale, bicycle, hair brush etc.) in an image. For each possible category, a set of bounding boxes and detection confidences should be found. A sample image taken from the Imagenet large scale visual recognition (ILSVRC) 2013 detection challenge (Russakovsky et al., 2015) training set is shown in Figure 3.4; the squirrel class is one of 200 classes in this Imagenet challenge track, a subset of the 1000 classes found in the Imagenet recognition challenge.

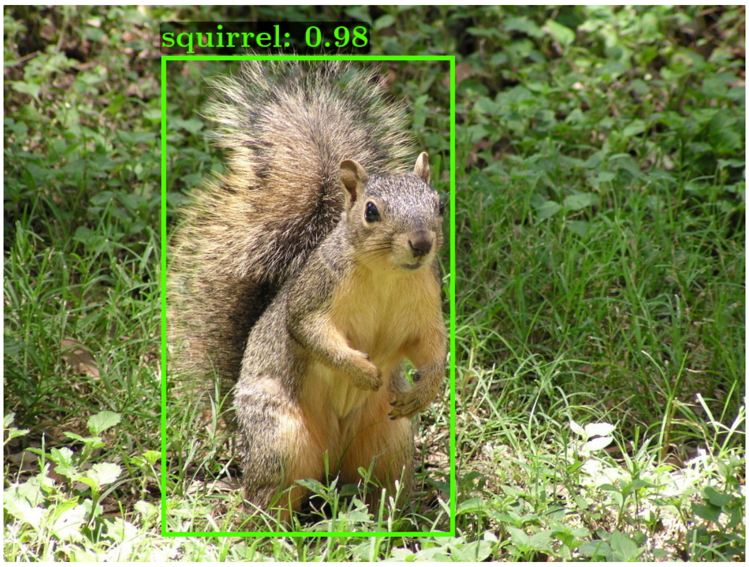

Figure 3.4: Sample object detection output with bounding box, class label and confidence.

Before the deep learning boom, or rather the renaissance of convolutional neural networks (CNNs), around 2012 most object detection methods built upon a two stage approach in which first low level features like Histogram of oriented gradients (HOG) (Dalal and Triggs, 2005) or Scale Invariant Feature Transform (SIFT) (Lowe, 2004) were extracted and then classified by *e.g.* a linear SVM. While those methods performed reasonably well, they involved a lot of hand crafting and their performance gains plateaued.

3 Related work

When Krizhevsky et al., 2012 proposed CNNs for Imagenet classification, they achieved a leap in classification performance: Top-1 error of 37.5% vs. 45.2% previously achieved as best result on the ILSVRC 2010 test set using SIFT features. Before that, most improvements were rather incremental in the 1-2% range. It was this massive improvement that sparked enormous interest in CNNs for almost any task in computer vision ever since. Object detection is certainly one of them and there are two different major streams of methods that have evolved up to this point. One stream of methods is based on first extracting possible candidate regions for objects and processing each region individually (R-CNN, Girshick et al., 2014), whereas others implement a single shot approach without region based processing (Liu et al., 2016; Redmon et al., 2016). Here, I want to quickly introduce the evolution of region based approaches which were consecutively published in major conference papers: R-CNN (Girshick et al., 2014), Fast R-CNN (Girshick, 2015), Faster R-CNN (Ren et al., 2015) and Mask R-CNN (He et al., 2017), because I apply Mask R-CNN in my own approach (see Section 4.2.2).

The original R-CNN approach extracts candidate regions using selective search (Uijlings et al., 2013). Those regions of interest (RoIs) are any areas of an image that might contain an object, *i.e.*, foreground areas. Thus, any region proposal method is usually class agnostic. For each of the RoIs, R-CNN extracts a feature map using a CNN backbone and then assigns a class label using linear SVM. Since a CNN forward pass is needed for any of the (*e.g.* 2000) RoIs, this approach is rather inefficient and was thus later replaced by Fast R-CNN (Girshick, 2015).

The major contributions of Fast R-CNN are the RoI Pooling layer and shared computation of the convolutional feature maps. For each image, only one forward pass of the backbone network is necessary to compute a single global feature map for the whole image. Afterwards, the *RoI Pooling* layer locally aggregates the features from the global feature map within each RoI. Finally, for each RoI a softmax classifier is used for class labelling and a bounding box regression network refines the bounding box borders of the RoI. Both of those networks are lightweight fully connected architectures. Due to the shared computation and joint training, this architecture is significantly faster (up to $213\times$ for inference and $18\times$ for training) and also more accurate ($+3\%$ mAP on VOC 2012 (Everingham et al., 2012)).

Faster R-CNN (Ren et al., 2015) seeks to remedy one of the major downfalls of Fast R-CNN which is that it still uses pre-computed RoIs, which have to be stored to disk and loaded for training and testing. To achieve this, they introduced Region Proposal Networks (RPNs), which are light weight convolutional architectures that also operate on the feature map obtained from the backbone network. RPNs use a sliding window approach, where a small spatial window is slid over the feature map and for each window bounding boxes and objectness scores are regressed. Thus, more computation is shared, since the RoI proposals are calculated from the same feature map that is used for the detection itself (*i.e.* the Fast R-CNN head). The authors report an increase in performance by $+2\%$ mAP on VOC 2012 and a speedup of up to $34\times$. R-FCN (Dai et al., 2016) achieves similar performance with another $2.5 - 20\times$ speedup using a fully convolutional architecture and hence more shared computations.

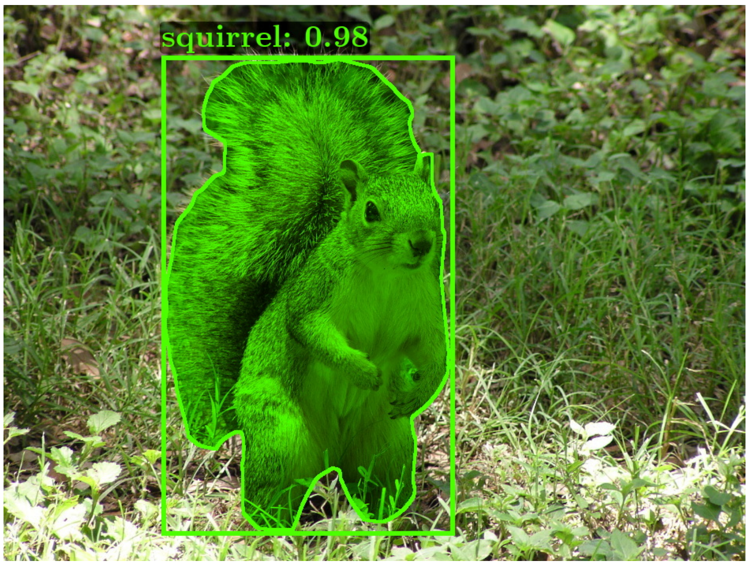

Figure 3.5: Sample instance segmentation output with bounding box, class label, confidence and segmentation mask.

Finally, the most recent Mask R-CNN (MRCNN) (He et al., 2017) extends the previous R-CNN based object detection approaches by instance aware semantic segmentation. In semantic segmentation the aim is to provide a binary segmentation mask in addition to a bounding box for each instance of each category. The binary segmentation mask encodes which pixels represent the object and which do not. Fundamentally, MRCNN replaces *RoI Pooling* with *RoIAlign* and adds a mask regression head to Faster R-CNN. *RoIAlign* avoids quantization of RoI boundaries by avoiding rounding operations when mapping RoIs to feature map coordinates and subdividing RoIs into a regularized grid. This is achieved by the use of bilinear interpolation instead of a rounding operation, which in the end results in higher alignment accuracy. The *mask head* is a fully convolutional neural network that outputs one binary segmentation mask for each class and each RoI and is trained on an average binary cross-entropy loss. He et al., 2017 report relative improvements of $\sim 30-50\%$ on the Microsoft COCO (Lin, Maire, et al., 2014) test and val sets compared to the winners of the COCO 2015 and 2016 segmentation challenges.

3.5 Alternative solutions to the *CVPR'17 Velocity Estimation Challenge*

Out of the contestants in the *CVPR'17 Velocity Estimation Challenge*, only the top 3 performing teams were asked to submit details on the method they used to tackle the challenge, and the winner was invited to give an oral presentation at the Autonomous Driving Workshop at CVPR'17. The challenge leaderboard top 3 is shown in Table 3.1. Our approach outperformed the second placed entry by 15% and a post deadline submission using only tracking features, but more careful model selection, increased the margin to 26%. For the entry that placed third, no further information on the methods used is available, but the methods that achieved second place are known [2].

	E_V	$E_{V,near}$	$E_{V,med}$	$E_{V,far}$
Ours tracking*	**1.25**	**0.12**	**0.54**	3.11
Ours	**1.30**	**0.18**	**0.66**	**3.07**
Rank2 team	1.50	0.25	0.75	3.50
Rank3 team	2.90	0.55	2.21	5.94

Table 3.1: Challenge leaderbord top 3, ranked by overall velocity mean squared error E_V given in $\frac{m^2}{s^2}$ (see Section 5.2 for details). *submitted post deadline

Second place approach The second place entry, submitted by Thomasz Wrona, employs a two-stage method that relies on first extracting the tracks for all vehicles and then regressing velocity values for each sequence of bounding boxes. As a

[2] `http://benchmark.tusimple.ai/static/files/poster_velocity_2.pdf`

tracker he uses *Median Flow* (Kalal et al., 2010), which is the same method as used in this thesis. In contrast to our approach, Wrona does not mention any substitutions necessary for failed tracks, whereas we substitute missing tracks with bounding boxes obtained by an *MIL* tracker (Babenko et al., 2009). Additionally, Thomasz Wrona applies a data augmentation scheme, where he horizontally flips bounding boxes and reverses sequences. For the training of the regression stage, he uses a shallow 2 hidden layer (64 and 16 units) feed forward neural network with tanh activations that is trained for 5000 iterations on a velocity mean squared error (MSE) using ADAM (Kingma and Ba, 2014) and Adagrad (Duchi et al., 2011) optimizers. Wrona finds that using all 40 frames performs better than skipping all but 5 key frames. While the general idea of employing a two-stage feature extraction and regression approach is basically identical to our method, some implementation details vary, such as the trackers used and the neural network setup as well as training regime.

4 Vehicle Velocity Estimation

This chapter presents the methods I developed and employed to tackle the *CVPR'17 Velocity Estimation Challenge*. The objective of the task given is to estimate the relative velocity as well as the position of given vehicles seen in short dashcam video snippets.

First, I establish a baseline to compare the approach to. I describe the original baseline provided for the challenge and I introduce an own baseline using only simple geometry.

Second, I introduce the full approach, which aims to leverage the power of current machine learning techniques in two separate stages. In the feature extraction stage, features beneficial to the task are extracted and fed forward to a light-weight Multilayer Perceptron (MLP) regression stage, working on these features to regress velocities and positions of vehicle instances.

4.1 Establishing a baseline

In order to evaluate the meaningfulness of the approach and to benchmark its performance, it is necessary to first set a baseline performance. The first approach was provided by the challenge organizers and is a naive baseline assuming 0 m/s velocity for all vehicles. The other baseline approaches exploit triangle similarities by means of simple geometry.

4.1.1 Challenge baseline

The original challenge baseline is the trivial solution to the problem, which simply assumes 0 m/s relative velocity for each vehicle. While this might sound irrational at first, it actually makes sense considering freeway traffic scenarios. On highways and freeways the traffic flow is generally assumed to be most efficient and safe in the case that all vehicles keep a constant distance to other vehicles. This also implies a constant relative velocity, at least within the same lane as the reference vehicle. A difference in relative velocity will most often occur in the course of passing maneuvers, although the differences in relative velocity can still be assumed to be small. Consequently, the assumption of 0 m/s relative velocity of all surrounding vehicles is reasonable yet leaves room for improvement.

4.1.2 Geometry-based estimation

As mentioned in Chapter 2, in addition to the RGB, velocity and position data, the dataset also includes camera calibration parameters. One of the known values is the intrinsic matrix in the form

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 714.15 & 0 & 675.58 \\ 0 & 710.37 & 376.26 \\ 0 & 0 & 1 \end{bmatrix} \text{px}, \tag{4.1}$$

where f_x and f_y are the horizontal and vertical focal lengths and c_x and c_y are the coordinates of the principal point. Note that the calibration matrix is given in image coordinates and thus the unit is pixels. The other known calibration parameter is the height above ground of the camera $H = 1.80$m. A calibrated camera can be used to measure the angle between two lines of sight. This allows for the distance between the vehicle and the camera to be measured by exploiting simple triangle similarities. The temporal evolution of the distance gives a vehicle's velocity. In the *distance from width* approach an average vehicle width has to be assumed, while in the *distance from height* approach the knowledge of the camera height is employed.

Distance from width

This method assumes that all vehicles are of exactly the same width, which is a constant that needs to be chosen with care. Additionally, to achieve valid results, tight bounding boxes that only include the rear of the vehicle are necessary; if a vehicle is not directly in front of the camera, some of the side is visible as well. Figure 4.1 depicts the fundamental idea of this approach which relies on similarity between the green and blue triangles. The blue triangle consists of the width of the vehicle W as well as the distance from vehicle to camera along the driving direction d_X, both in vehicle coordinates (meters). The green triangle consists of the horizontal focal length f_x and the width of the vehicle w in the image plane I, both in image coordinates (pixels). Since the width of vehicles W in the dataset is not known, an assumption of an average width is required. W is chosen such that it minimizes the velocity estimation error, which results in W=1.82 m for tracked bounding boxes and W=1.27 m for MRCNN bounding boxes (see Section 5.3.2 for details). w is the width of the vehicle's bounding box and f_x is known from the calibration matrix **K** from (4.1).

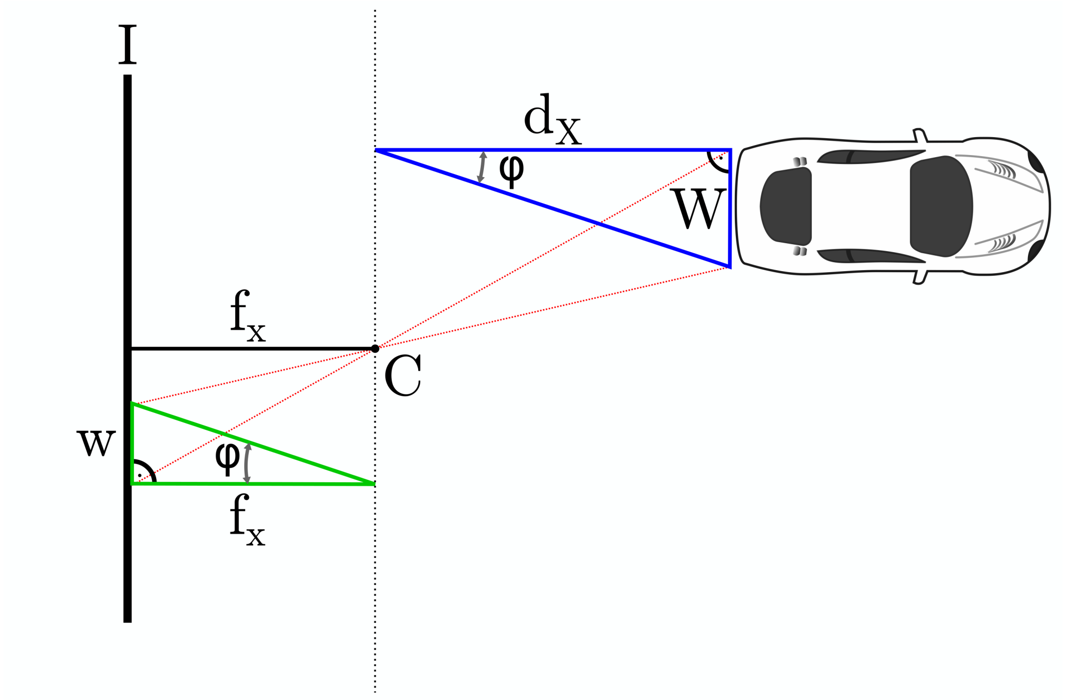

Figure 4.1: Schematic depiction of inferring distance d_X from vehicle width. Assuming a fixed width w the distance d_X can be calculated from triangle similarities.

Since the angle φ is the same for both triangles,

$$\frac{d_X}{W} = \frac{f_x}{w} \tag{4.2}$$

holds which in turn gives the desired distance d_X:

$$d_X = \frac{f_x W}{w}. \tag{4.3}$$

Distance from height

In contrast to the previously described approach, the following in theory does not rely on any assumptions, given that the optical axis is parallel to the ground. However, the pitch of the camera is undisclosed and thus an approximation is necessary. The approximation consists of shifting the principal point c_y such that the velocity estimation error is minimized, which results in $c'_y = 329$ px for tracked bounding boxes and $c'_y = 327$ px for MRCNN bounding boxes (see Section 5.3.3 for more details). Figure 4.2 shows the basic geometric constellation which is used for calculating the distance d_X. Again the blue triangle, which is in this case spanned by the distance d_X and the height of the camera H, is given in vehicle coordinates. The green triangle, spanned by the vertical focal length f_y and the distance from camera center to the lower bounding box border $h = |p - c'_y|$, both in pixels. f_y and h are known parameters from the calibration, as described in Chapter 2. c'_y is the 'virtual horizon'. This value indicates the average location of the horizon within the camera image, which depends on the angle of the camera and any variations thereof.

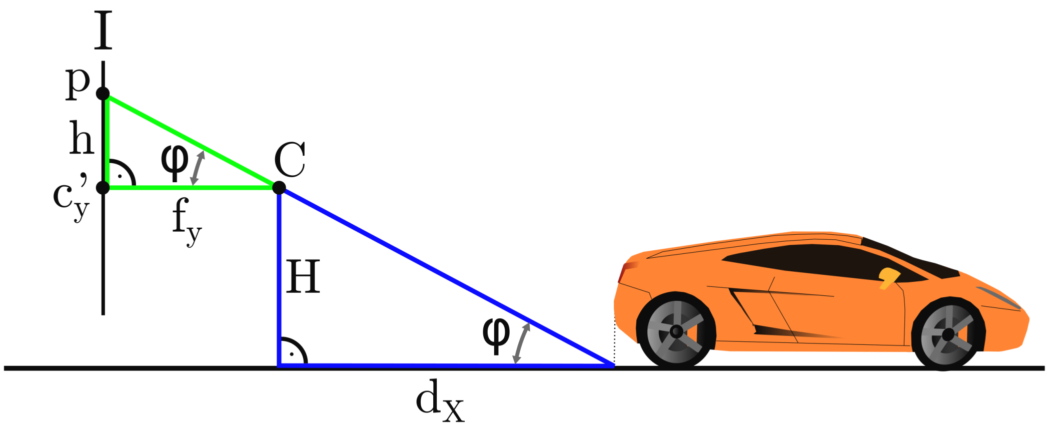

Figure 4.2: Schematic depiction of inferring distance d_X from y coordinates in the image. The known height above ground of the camera is used to establish a triangle similarity for calculating d_X.

Both triangles are similar since they are rectangular and share the same common angle φ. Hence

$$\frac{d_X}{H} = \frac{f_y}{h} \tag{4.4}$$

holds and d_X can be computed:

$$d_X = \frac{f_y\, H}{h}. \tag{4.5}$$

While this approach on the one hand comes with the benefit of theoretically being well defined, on the other hand it comes with the downside of being potentially unstable. This is caused by the distance h which can approach zero, in which case (4.5) diverges. Any deviation of the horizon from the 'virtual horizon' will cause this behavior. Apart from carefully choosing c'_y (see Section 5.3.3) another constraint imposed is clamping any values of $d_X < 0.1$ to $d_X = 0.1$. Practically speaking, h depends on the angle of the camera and the surface geometry of the road. The angle of the camera is influenced by the general road condition, since any bumps will temporarily alter the pitch of the vehicle, and thus the angle (or pitch) of the camera. This problem could be alleviated by stabilizing the camera image in either software or better yet hardware with *e.g.* a gimbal. Furthermore, any curvature in the incline of the road will shift the horizon, and will in turn cause a variation of h that is not only unrelated to the distance but might also cause an instability in the estimate.

Lateral distance

Both approaches have thus far only considered longitudinal distances. While they differ in their method of calculating the distance d_X, they share the calculation of the lateral displacement d_Y. With the distance d_X given, the lateral displacement d_Y can be computed following the same method of finding triangle similarities as before, which is shown in Figure 4.3. The blue triangle in vehicle coordinates is spanned by the longitudinal and lateral distances d_X and d_Y respectively. The green triangle in image coordinates is spanned by f_y and $d_y = p - c'_x$, where c'_x represents the origin of the vehicle coordinate $y = 0$ in image coordinates and is not to be confused with the principal point coordinate c_x.

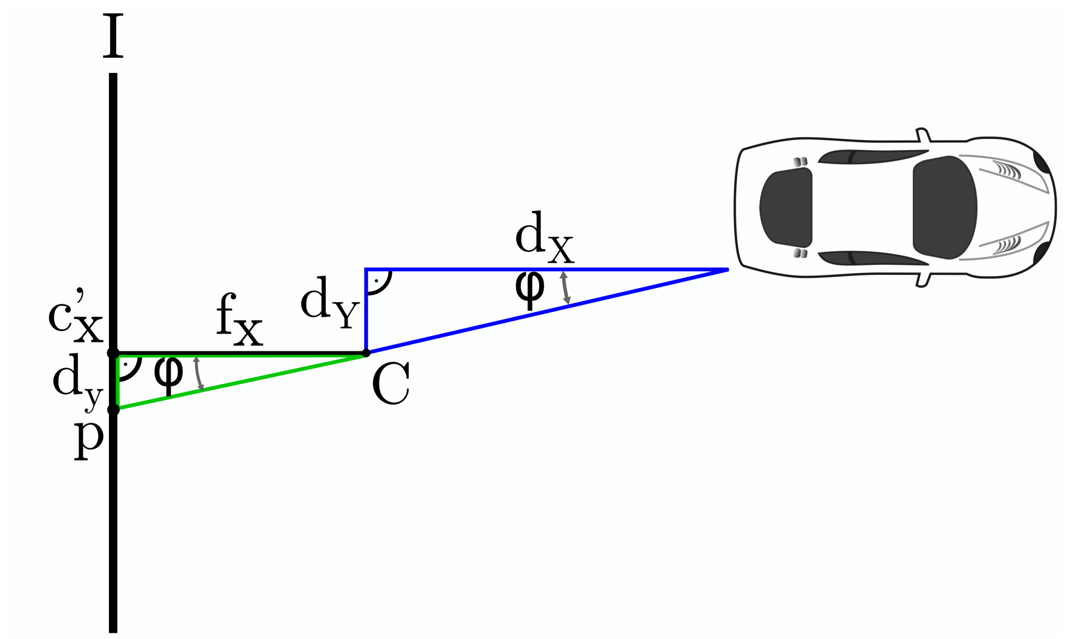

Figure 4.3: Schematic depiction of inferring lateral displacement d_Y from known longitudinal d_X distance. d_Y is obtained using triangle similarities.

Again triangle similarities can be exploited since the angle φ is the same in both triangles.

$$\frac{d_Y}{d_X} = \frac{d_y}{f_x} \tag{4.6}$$

holds and thus follows

$$d_Y = \frac{d_y \, d_X}{f_x}. \tag{4.7}$$

Derivation of c'_x

So far, the derivation of c'_x has not been mentioned. A derivation of c'_x is necessary because the camera is apparently not mounted centrally on the vehicle, which results in an offset between the principal point c_x and the origin of the vehicle coordinate system ($Y = 0$). This offset is visible in Figure 4.4, where the red vertical line represents the c_x and the vehicle enclosed by the green bounding box is at $d_Y = 0.05$ m (*i.e.* approximately in the center). Since the baseline methods require the world coordinate $Y = 0$ to be in line with the principal point, the principal point has to be shifted to c'_x.

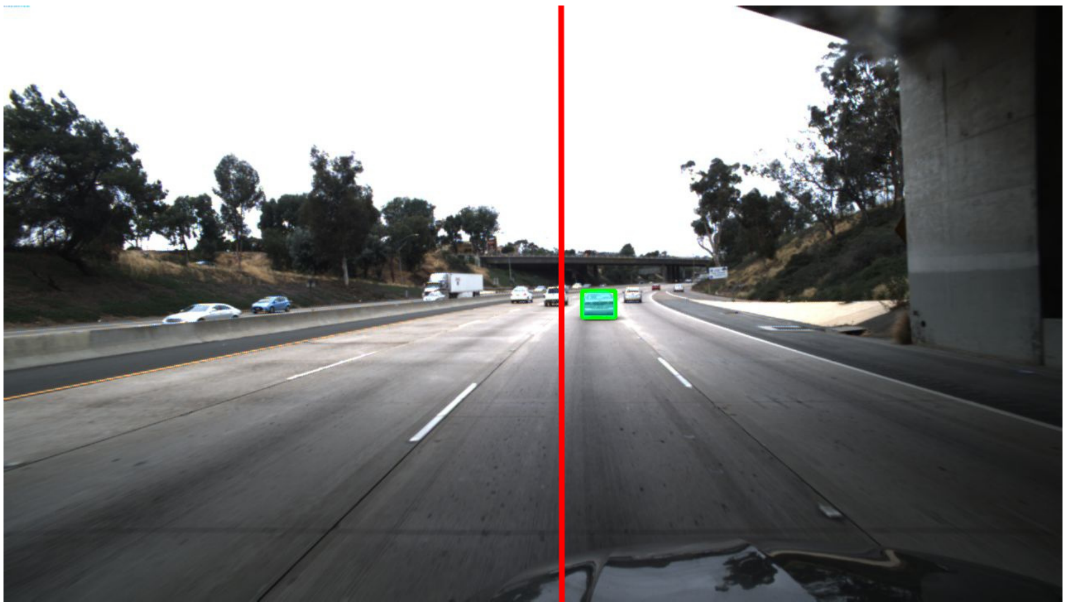

Figure 4.4: x=c_x (red line) and a vehicle at $d_Y = 0.05$ m (green bounding box). The vehicle coordinate center is not in line with the principal point, since the camera is not mounted centrally on the vehicle or panned accordingly.

The ground truth position data (d_X and d_Y) can be used to calculate an average for c'_x using

$$d_{x,px}(n) = \frac{d_Y(n)\, f_x}{d_X(n)} , \tag{4.8}$$

where n is a single vehicle. $d_{x,px}$ is the lateral displacement equivalent to d_Y in pixels. Since the ground truth position is given relative to the point of each vehicle which is closest to the observer, two separate cases need to be distinguished:

$$c'_x(n) = \begin{cases} X_{max}(n) + d_{x,px}(n), & d_Y(n) \ < \ 0 \\ X_{min}(n) + d_{x,px}(n), & d_Y(n) \ \geq \ 0 \end{cases} \tag{4.9}$$

An average over the number of vehicles N yields the final result

$$c'_x = \frac{1}{N} \sum_{n=1}^{N} c'_x(n) = 713.85 \text{ px}. \tag{4.10}$$

Velocity

With known distances at different timesteps, the velocity of the vehicle is defined as

$$\mathbf{v}(t) = \frac{\Delta \mathbf{d}}{\Delta t}, \tag{4.11}$$

where $\mathbf{v}$ and $\mathbf{d}$ are two dimensional in vehicle coordinates X,Y and

$$\Delta \mathbf{d} = \mathbf{d}(t_2) - \mathbf{d}(t_1). \tag{4.12}$$

Δt depends on how many frames are skipped between t_1 and t_2. Let k be the number of frames skipped and τ be the interval between two consecutive frames (in this case $\tau = 2/40$ s — 40 frames in 2 seconds, see Chapter 2), then

$$\Delta t = k\, \tau. \tag{4.13}$$

Substituting (4.12) and (4.13) into (4.11) yields

$$\mathbf{v}(t) = \frac{\mathbf{d}(t_2) - \mathbf{d}(t_1)}{k\, \tau}. \tag{4.14}$$

Note that as a bound for added stability, the calculated relative velocities are clipped at 25 m/s.

A certain frame skip always considers all possible combinations within the 40 available frames. For example, assuming a frame skip of $k = 5$, the pairs of frames that would be used for calculation are

$$\{[1,6],\ [2,7],\ [3,8]\ \dots\ [33,38],\ [34,39],\ [35,40]\},$$

and thus all of the available data is used. For each of the individual pairs a velocity error (see Section 5.2) is calculated and the median of the intermediate results yields the final result.

The geometry based estimation methods are evaluated in Sections 5.3.2 and 5.3.3.

4.2 Neural network based approach

The original idea for the entry in the *CVPR'17 Velocity Estimation Challenge* was to extract features that might be beneficial inputs for a regression algorithm that should learn to estimate velocities based on those high level features. An overview of the overall idea is given in Figure 4.5. First, vehicles are tracked through the unannotated frames. Next, optical flow and disparities are calculated at the vehicle positions. Those features are then stacked to a single input vector for a multilayer perceptron that is trained to output velocity and position values.

Since the only available input data are RGB frames, an extraction of meaningful representations is essential. Theoretically, a regression of vehicle velocities directly from RGB frames is conceivable; however, it would require training of a large convolutional neural network which in turn requires a considerable amount of labeled training data (*i.e.* orders of magnitude larger than available in this case).

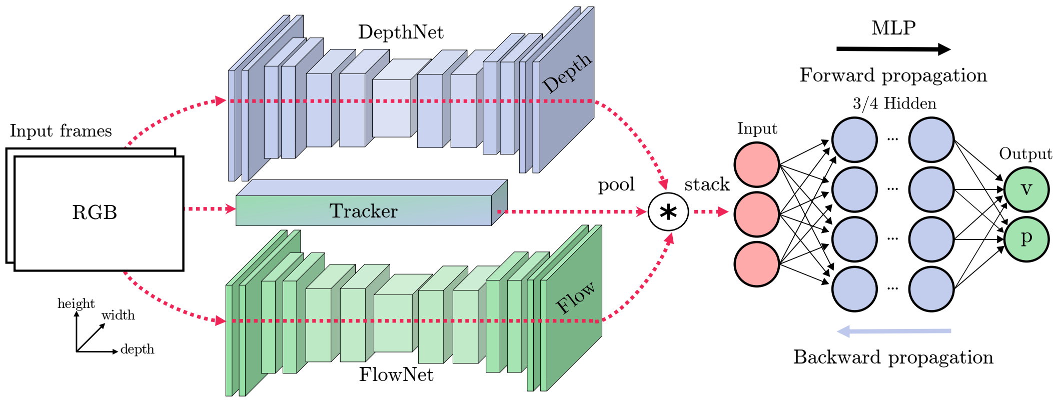

Figure 4.5: Overview of the overall architecture for the neural network based approach. First a feature extraction stage calculates features from RGB frames that are locally aggregated and stacked as input for a fully connected regression stage that outputs velocity and position vectors.

To estimate vehicle velocity, the raw RGB video data is first transformed into a feature space that benefits the task. I decided upon three types of complementary features that theoretically allow a well defined solution of the problem. First, tracks (*i.e.* trajectories of the 2D object bounding box and segmentation mask over time) serve as a basis to preserve the location and identity of vehicle across each sequence. Second, depth (*i.e.* disparity estimates from monocular images) gives information on how far each pixel is away from the camera. Third, motion (*i.e.* optical flow estimates between consecutive frames) yields pixel-level displacement vectors between temporally adherent frames. The temporal evolution of depth and motion together encodes the 3D motion of the scene, and consequently allows to estimate the velocity vector of each vehicle using the tracked bounding boxes. The remainder of this section describes how each one of these features is extracted.

4.2.1 Tracking

For a given vehicle defined by a bounding box in a single frame (*i.e.* the last frame of each sequence), tracking over the temporal extent of the input subserves all further processing steps. Since the vehicles are labeled solely in the last frame, tracking needs to go backwards in time. Figure 4.6 gives an example, where the vehicle enclosed by the green bounding box is tracked backwards through time (right to left); the blue bounding boxes are obtained by the tracker. The tracking task as such does not pose any major difficulties on this dataset, since there is little movement in between frames, occlusions only rarely occur, and all annotated vehicles are visible throughout the sequence (*i.e.* they do not leave the frame).

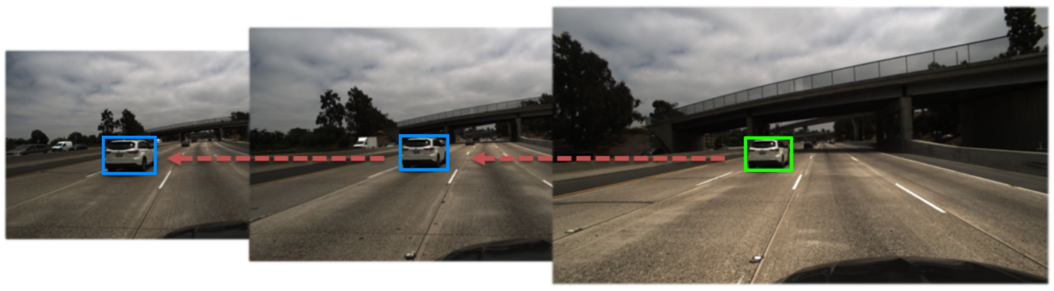
Figure 4.6: Tracking the vehicle in the rightmost frame (green bounding box, known) backwards through time (blue bounding boxes are calculated).

A variety of well established tracking algorithms are readily available in literature. According to the relatively low inherent difficulty of the tracking task, a relatively lightweight architecture can be employed. Still, the baseline methods as well as the neural network approach rely on tight bounding boxes (*i.e.* the outline around the object should not leave any large margins). Thus, my first choice is the *Median Flow* (Kalal et al., 2010) tracker, since it is able to resize the bounding box throughout the sequence and therefore can account for changes in apparent size of objects. In some cases, however, *Median Flow* tracks fail due to occlusions, in which case bounding boxes obtained with an *MIL* (Babenko et al., 2009) tracker are substituted. The *MIL* method is more robust, but does not track tightly. I use implementations of both algorithms from the `OpenCV` library (Bradski, 2000), with all default settings. Both perform well out of the box with no further adjustments required.

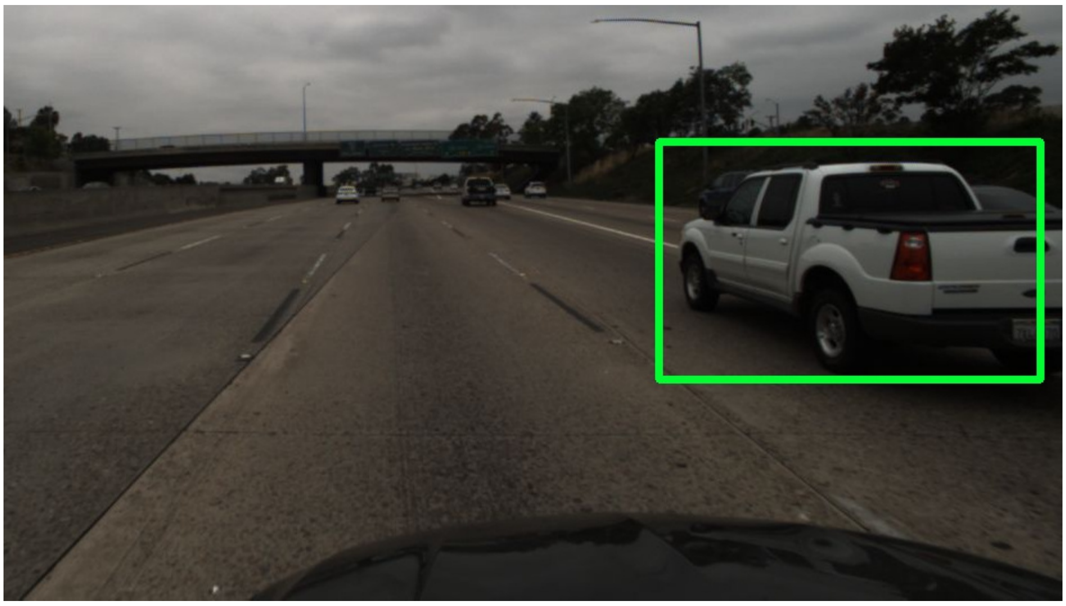

Figure 4.7: Example of an imprecisely annotated bounding box. The ground truth bounding box does not tightly enclose the vehicle.

On a final note, it should be mentioned that some of the ground truth annotations are neither very accurate, nor tight. In order to obtain tight tracks with the tracking algorithms used, a tight reference bounding box in the starting frame is crucial. An example of an inaccurate annotation is shown in Figure 4.7. Clearly the vehicle is not tightly enclosed by the bounding box on the left, right and top borders. In addition to complicating the tracking task with respect to tightness, this problem could also potentially lead to tracking confusions in case of partial occlusions. Cases as the one in Figure 4.7 are rare, but can not be ruled out, especially in near range.

4.2.2 Mask R-CNN tracking and segmentation masks

Mask R-CNN (He et al., 2017), or MRCNN, is a simple yet powerful framework that simultaneously computes object bounding boxes and instance aware segmentation masks. Conceptually MRCNN is a derivative of Faster R-CNN (Ren et al., 2015), which replaces *RoIPool* (Girshick, 2015) with *RoIAlign* and adds a *mask head* to the second stage of Faster R-CNN (see Section 3.4 for details). MRCNN, originally trained on the Microsoft COCO (Lin, Maire, et al., 2014) dataset, is capable of producing output on the velocity estimation dataset as shown in Figure 4.8 without any further in-domain optimization. The segmentation masks inferred by MRCNN are surprisingly accurate for near to medium ranges. Note how *e.g.* the hood of the observer car is accurately segmented and labeled as a car with 99% confidence.

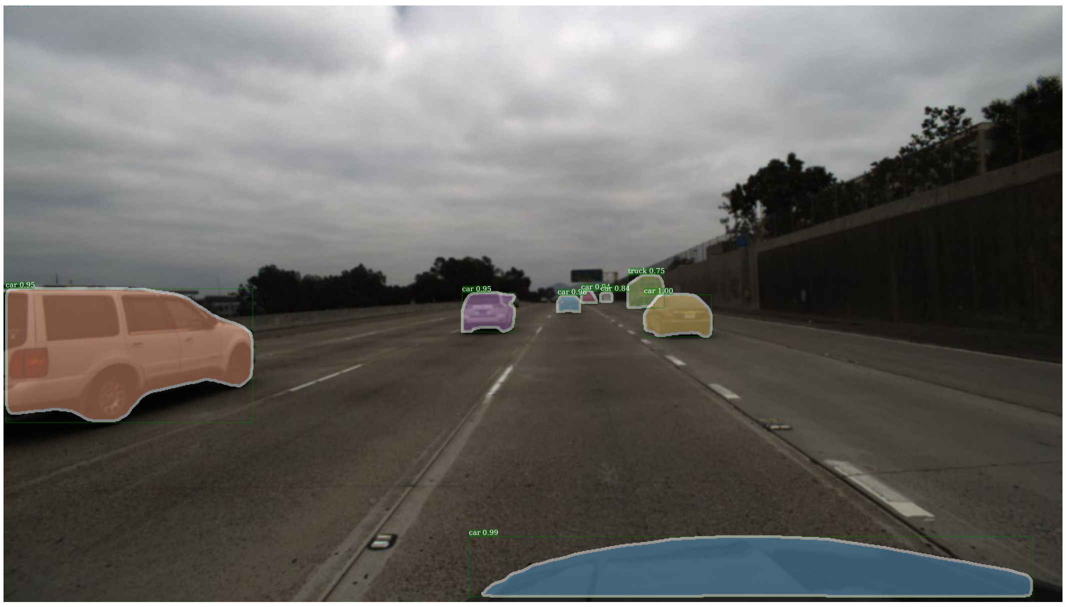

Figure 4.8: Sample output of MRCNN segmentation masks, class labels and confidences.

MRCNN masks and bounding boxes are obtained using the model configuration denoted `X-101-64x4d-FPN` in He et al., 2017, which uses a 101 layer ResNeXt (S. Xie et al., 2017) backbone architecture with 64 groups of width 4 and a Feature Pyramid Network (Lin, Dollár, et al., 2017). This is an architecture with high predictive power at various scales. It is trained on the Microsoft COCO (Lin, Maire, et al., 2014) dataset, which provides over 120k images with over 880k instance annotations for 80 different classes.

Segmentation masks come with the benefit of pixel level accuracy and thus with implicitly tight bounding boxes; the bounding boxes are chosen such that the segmentation mask is tightly included within the box. While the segmentation masks are instance aware, which means that they distinguish between separate vehicles rather than providing one mask for all vehicles, the identity of instances is not preserved over the temporal extent. To track masks and their bounding boxes backwards through the frames like in Section 4.2.1, a simple scheme based on Intersection over Union (IoU) is used:

$$\mathrm{IoU} = \frac{A(\mathrm{box_a} \cap \mathrm{box_b})}{A(\mathrm{box_a} \cup \mathrm{box_b})} \tag{4.15}$$

IoU is the area of intersection over the area of union of two bounding boxes and it gives a measure of congruence, where IoU=1 means full overlap and IoU=0 is no overlap. An example of two squares with IoU=0.4 is given in Figure 4.9.

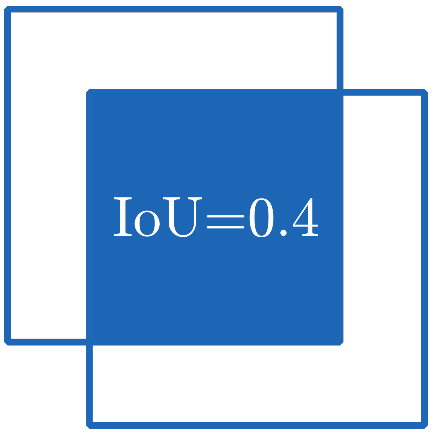

Figure 4.9: Two squares with IoU=0.4 .

For the tracking task, IoUs for bounding boxes of vehicles in consecutive frames starting in the last frame are calculated. All vehicle bounding boxes (*i.e.* COCO car, bus and truck categories) with a confidence above 40% and an IoU above 0.4 are counted as a valid track. Choosing both the confidence and IoU thresholds rather conservatively is a compromise that is necessary because the accuracy of far range samples is limited and larger displacements between frames need to be accounted for. The problem with vehicles at larger distances is that the pixel level segmentation accuracy and detection confidence stagnates due to smaller spatial scales. The detection based tracking employed here also suffers from jitter of bounding boxes between frames since there is no smoothing employed over the temporal dimension by methods such as Kalman filtering, which leaves room for potential improvement. Whenever this tracking regime still fails due to one of the reasons above, the tracks from Section 4.2.1 are substituted as a fallback.

4.2.3 Dense depth map prediction

Depth is undoubtedly an instrumental feature for the task, since a reliable depth map readily provides the solution for velocity in the X direction. Unfortunately, for monocular imagery, depth map prediction is an ill-posed problem, which is not as easily solved as in the case of stereo image pairs. Even for humans it is quite challenging to estimate distances with one eye covered, but still they are able to perform reasonably well, which requires experience. This is where recent deep learning architectures can show their strengths, since through thorough training they are able to acquire a similar grade of experience by learning from a multitude of samples.

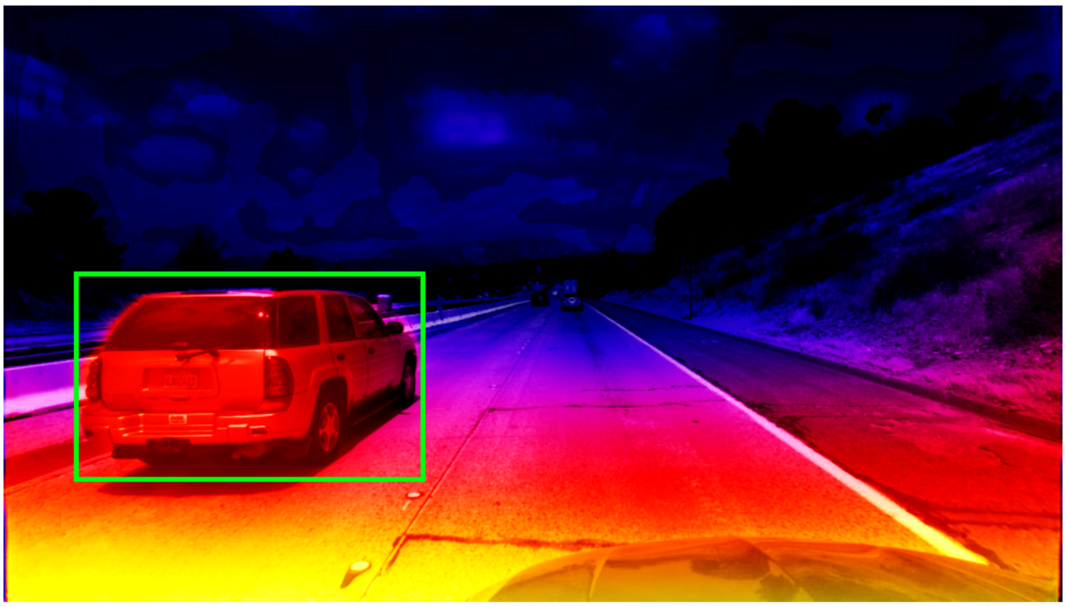

Figure 4.10: Sample frame with dense depth map overlay. Brighter/warmer tones indicate smaller distance. The approach clearly separates the vehicle from the background.

A recently described deep architecture (Godard et al., 2017) that learns monocular depth map prediction via novel view synthesis in a stereo environment serves to predict dense depth maps. This is achieved by synthesizing the image from one camera from the other (*e.g.* left from right), where one of the images is used as a supervision signal. The warping operation that is thus learned implicitly carries information on the disparities in the source image. At the time of implementation, this was the most powerful architecture for the task that was publicly available.

The model employed here is pre-trained on *KITTI* (Geiger et al., 2012) and *Cityscapes* (Cordts et al., 2016) stereo images for predicting the dense disparity maps (note that this model is trained without disparity ground-truth). The model is provided by the authors and publicly available on github[1]. The model used here is called `model_city2kitti`, which is pre-trained on *Cityscapes* and fine-tuned on *KITTI*. It was found to generalize best by the authors, which is why I decided to employ this model as well.

A sample output on a frame of the velocity estimation dataset is shown in Figure 4.10. Visually, the results obtained appear valid, at least for close range, where the vehicle depicted by the green bounding box is clearly separated from the more distant background. Note that larger distances correspond to darker/colder tones in the color coding used. For farther distances, the capabilities of the algorithm are limited by the small sizes of objects. Since the models are trained on inputs of size 512x256px, the input RGB images need to be resized before inference for optimal performance and also faster inference times. Moreover, the dense depth maps are spatially pooled in a later stage, which is why a high pixel level resolution is of no real benefit to the approach.

4.2.4 Dense motion prediction

Another state-of-the-art neural network architecture, *FlowNet2* (Ilg et al., 2017), is used to retrieve motion information by extracting dense optical flow maps. *FlowNet2* treats optical flow estimation as a supervised learning problem, where a convolutional neural network is trained on a volume of two stacked input frames with ground truth optical flow as a supervision signal. Since recovering ground truth optical flow data poses a major difficulty in the case of real world data, synthetic data is used most often. I employ a *FlowNet2* architecture trained on the synthetic *FlyingChairs* (Dosovitskiy, Fischer, Ilg, Häusser, et al., 2015) and *FlyingThings3D* (Mayer et al., 2016) datasets. The framework yields 39 dense u, v flow maps from 512x256px input images (analogous to the input size used for the depth estimation). The output is 39 flow maps, as opposed to the input of 40 frames, since optical flow always has to be calculated between a current frame and a reference frame, which leaves 39 unique pairs within 40 frames.

[1]`https://github.com/mrharicot/monodepth`

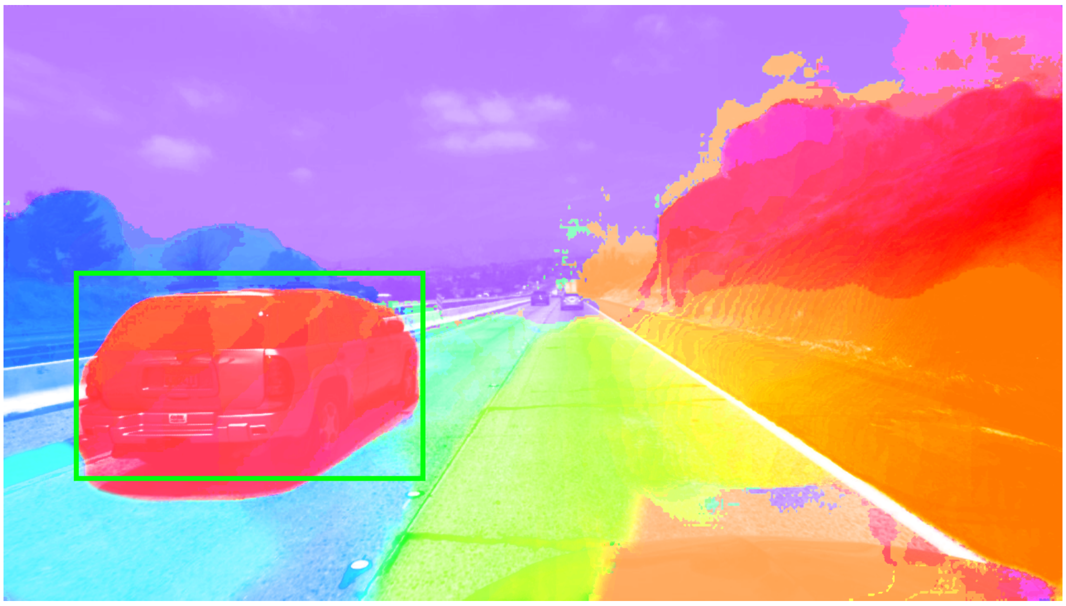

Figure 4.11: Sample frame with optical flow prediction overlay. Middlebury (Baker et al., 2011) like flow encoding; hue indicates different directions and intensity different magnitudes of movement.

The code[2] used for inference has been made publicly available by the authors. Note that at the time of my implementation the only publicly available models were ones trained on *FlyingChairs* and *FlyingThings3D*. In the meantime models trained on the *KITTI* and *sintel* (Butler et al., 2012) datasets have been released additionally. Using models trained on *KITTI* might further increase performance due to in domain knowledge acquired by the architecture, which remains left for investigation in future work.

`FlowNet2`, which is the full and best performing architecture described in Ilg et al., 2017, is used. A sample of an extracted optical flow feature map is shown in Figure 4.11. Again the architecture infers a visually sound representation limited to close distances. For larger distances, foreground pixels are no longer separated from background and the estimation becomes inaccurate. Here again the convolutional neural network architecture is limited by small pixel displacements between frames. A larger temporal stride could potentially improve this shortcoming, but lies beyond the scope of this thesis.

[2]`https://github.com/lmb-freiburg/flownet2`

4.2.5 Transformation into feature space

An MLP architecture expects a fixed size input vector and thus the extracted features need to be pre-processed before they can be fed forward to the regression stage. The depth and motion cues are computed globally as dense per pixel features and further processed by locally aggregating the dense predictions within the bounding boxes established by the tracking stage. Note that first the bounding boxes have to be resized from the native 1280x720px size to the input/output size of the deep feature extraction stages of 512x256px. The local aggregation consists of first shrinking the box by 10% in width and height and then average pooling over the dense feature maps within each re-scaled tracked bounding box. Shrinking the bounding box reduces the variance of the average, since flow and depth cues tend to be inaccurate at the object boundaries and less or no background lies within the smaller bounding boxes. Subsequently, the aggregated feature vectors are temporally smoothed using a Gaussian kernel of width $\sigma^2 = 5$, which is chosen in correspondence to the frame skip of 5 in the learning stage (see Section 4.2.7). For optical flow, this procedure is carried out individually for both the horizontal and vertical component x, y. The temporal smoothing provides robustness to short-term deviations of the camera orientation, that could be caused by movement of the vehicle due to road bumps and suspension wobble.

This local aggregation allows for a single float value for depth and two single float values for motion per vehicle and frame, drastically reducing the amount of data produced in the future extraction stage. The temporal axis, *i.e.* the sequence of features for each frame, is simply flattened to a single vector serving as input for the regression stage.

The pre-computed features allow for the use of relatively shallow fully-connected neural networks (Multilayer Perceptrons, or MLPs in short). This is especially advantageous for the task at hand, in which the number of given training examples is relatively small and learning can take advantage of high level abstract features. A comparatively simple, rather small and thus efficient 4-layer MLP architecture is sufficient for regression from feature space to vehicle velocities.

4.2.6 Range Split

The relationship of the pre-computed features to the learning output is highly non-linear. Consider, for instance, the size of bounding boxes around other vehicles: it rapidly changes when the vehicle is accelerating or decelerating close to the camera, while remaining more or less constant if the vehicle is far away regardless of the velocity. To facilitate learning under those varying conditions, the idea is to split the dataset into three disjoint parts. The split criterion is the distance from the observer (near/medium/far), which are the same ranges as introduced in Chapter 2. Subsequently, a separate model for each of the distances is trained, with potentially varying parameters and topologies.

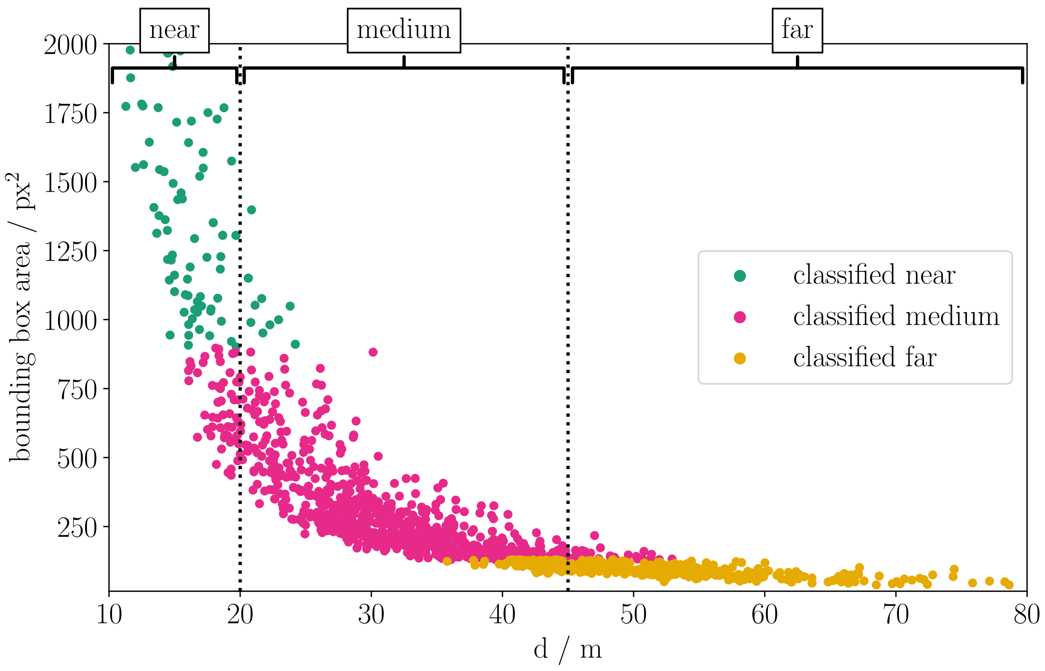

Figure 4.12: Split of ranges by bounding box area. The horizontal axis depicts the ground truth distance of each vehicle and the vertical axis is the corresponding area of the ground truth bounding box. Different classes represent the split into the three ranges using bounding box area.

Two different methods shall be compared here. The first method sets empirical boundaries to the area of vehicle bounding boxes. Anything above a bounding box area of 900 px^2 is considered near range, any area below 130 px^2 far range and all areas in between are medium range. This is the method originally used in our submission to the *CVPR'17 Velocity Estimation Challenge*. Figure 4.12 shows the split thus obtained, where the colors represent the predicted distance, and the boundaries of the actual ranges are represented by the dotted vertical lines. The plot is cropped for better visibility.

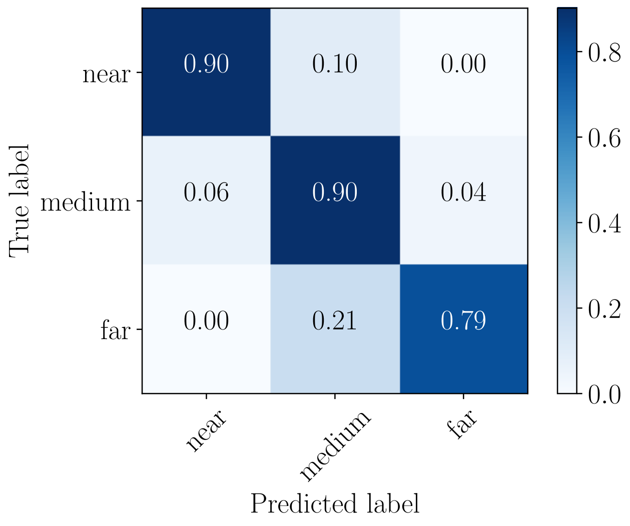

Figure 4.13: Confusion matrix of bounding box area range split. Near and medium ranges can be separated with little overlap, whereas for far range the overlap with the medium class is large (21% of far range samples fall into the medium category).

This method is clearly able to separate out the different ranges, but elicits some overlap in between them. This is also depicted in Figure 4.13, which is a visualization of the normalized confusion matrix of this simple hard decision boundary classification approach. In an ideal classifier, all elements on the main diagonal would be 1, and all other elements 0. Each row corresponds to the actual label of the class and each column to the predicted label. Hence, this approach is quite well suited for separating medium and close range samples. However, far range suffers from a higher overlap with the neighboring medium range class.

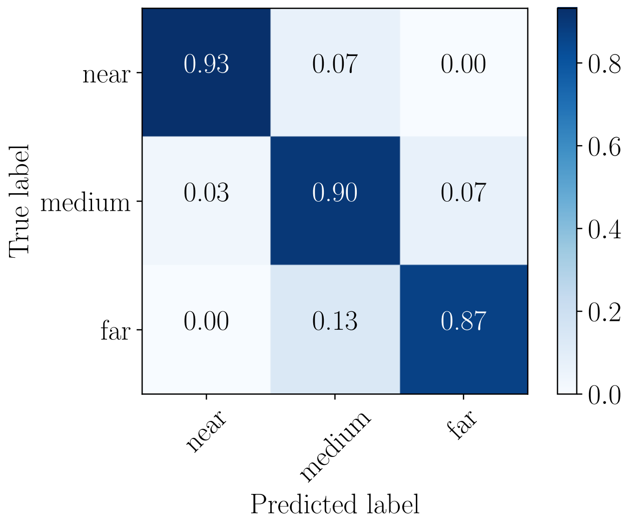

Figure 4.14: Confusion matrix of bounding box bottom coordinate range split. Compared to Figure 4.13 the overlap of near and especially far classes with the medium class is reduced.

A slight increase in classification performance can be achieved by considering the bounding box bottom coordinate instead of the area. This idea is similar to the distance from height baseline method introduced in Section 4.1.2. Here any bottom coordinate above 400.6 px is classified near range, any coordinate below 367.7 px far range and the rest medium range. Again, those are empirically chosen hard boundaries. The confusion matrix for this approach, shown in Figure 4.14, indicates a slight improvement in classification accuracy for the far and near classes. This is significant since those two classes are considerably smaller than the medium range class (see Chapter 2).

While both of these methods leave room for improvement, they are sufficiently accurate for the task at hand. Note that the range splits used in the ablation study in Section 5.4 use the available ground truth data for accurate splits. The range split described here only has to employed when no ground truth data is available, *i.e.* when inferring on unannotated test data.

4.2.7 Network Architecture

The regression stage is a simple, lightweight MLP architecture, the basic structure of which is shown in Figure 4.15. The number of inputs varies with the amount and types of input features, as well as the temporal stride that is used. The temporal stride was chosen to be k=5 in validation experiments. This means that only every kth frame is used for estimation. Notably, recycling the frames dropped by the temporal stride by stacking them into additional input vectors as separate samples did not improve performance. For example, when using all features, 8 values serve as input for a single frame. With a temporal stride of k=5 frames, this results in 64 input units. The inputs are fed to 3 or 4 hidden layers, each with 40, 60 or 70 neurons. The number of units per layer is a parameter that is examined in the ablation study in Section 5.4. For the challenge entry the following topologies (number of hidden layers $\times$ number of units per layer) were used for the individual ranges: 3×40 (near), 4×60 (medium), and 4×70 (far). The final output vector is of length 4 and consists of velocity and position values for both X and Y directions. It is generated by a linear output layer, as is common practice for regression tasks.

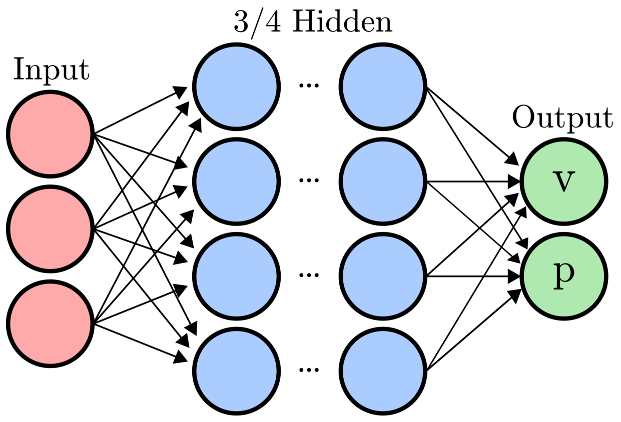

Figure 4.15: MLP structure used for velocity and position regression. A variable size input layer (dependent on the number of features and frames used) feeds 3-4 fixed size hidden layers and a 4 unit linear output layer provides velocity and position values.

The effect of various activation functions on the different subnetworks is investigated in Section 5.4. Originally, for the entry in the *CVPR'17 Velocity Estimation Challenge* concatenated rectified linear units (CReLU, Shang et al., 2016) were used as hidden units.

4.2.8 Network Training

Each individual network is trained on the MSE between network output and ground truth annotations for 2000 epochs using a minibatch size of 50. Notably, the training is carried out on velocity and position data, although evaluation only takes velocity estimation performance into account. Ground truth position values serve as an auxiliary learning target for the MLP regressor. For regularization, weight decay (l_2 regularization) of 10^{-5} and Dropout (Srivastava et al., 2014) of 0.2 are used. Also ADAM (Kingma and Ba, 2014) proved to be the most feasible optimization algorithm for the task at hand.

A partitioning scheme similar to k-fold cross-validation is used to exploit all of the available training samples. For each of the individual ranges, the samples are split into 5 partitions, where one is used for validation and the rest for training. After training for 2000 epochs on each separate combination of partitions the model with the lowest validation error is saved. In the end, this results in 3×5 models for the entire dataset. While this approach yields separate models specialized for each range, the number of examples per neural network is quite small (especially for near and far range), so care has to be taken with respect to overfitting. To counteract overfitting, in addition to Dropout and l_2 regularization, training is also stopped early, if the validation error does not improve after 500 epochs.

At inference time, a single output for every sample of each range is obtained by aggregating the feed forward outputs of each of the 3×5 models and averaging them. To split the samples into ranges with unknown ground truth, the split strategy introduced in Section 4.2.6 is used.

5 Experiments

This chapter gives an insight into the experiments that were conducted as well as the corresponding results. First, a split of the original training set (see Chapter 2) is established, that serves as a basis for experimental evaluation. Then, the baseline approaches (introduced in Section 4.1) are investigated and finally an ablation study on the various input features for the machine learning regression approach is presented.

5.1 Validation split

In order to be able to evaluate the performance of the different approaches, it is necessary to first split the available training data into a custom fixed training and validation set. This step is required since the original test set annotations are undisclosed. Instead of simply using a random split, I decided to take a more deterministic approach, taking into account the distribution of vehicles in near, mid and far range as well as the appearance of the individual samples. Here, appearance refers to the individual drives that the video snippets of the dataset are taken from. The problem here is that some of the samples in the training set are temporally consecutive and correlation in between such examples is very high. For a meaningful evaluation, the aim is to separate training and validation set for the upcoming experiments as clearly as possible.

This is achieved by first computing the 4096 dimensional *VGGNet fc2* feature vector (Simonyan and Zisserman, 2014). VGGNet is a convolutional neural network used for large scale image recognition, that has won the ImageNet (Russakovsky et al., 2015) 2014 challenge localization track. The network consists of a 16 layer convolutional block followed by a 3 layer fully connected head, adding up to 19 trainable layers with 144 million tunable parameters. The task of the convolutional block is to transform an input RGB image into a feature space that allows for classification into one of 1000 classes by the subsequent fully connected classification head. The fc2 features are the output of the last 'class agnostic' stage and provide a high level abstract encoding of the appearance of an image. The 1000 dimensional fc3 layer is already corresponding to the 1000 output classes.

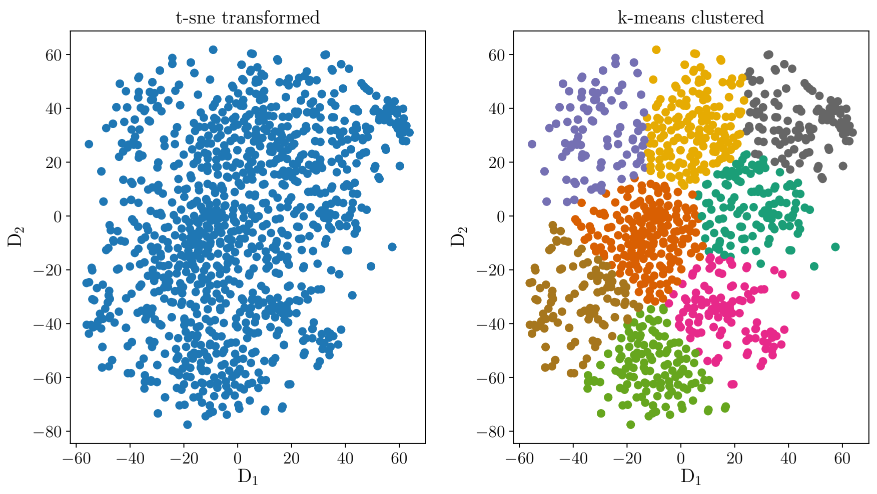

Figure 5.1: left: t-sne reduced 4096 dimensional fc2 feature vectors, right: k-means clustering of reduced data

Further, *t-sne* (Maaten and Hinton, 2008) is used to reduce the dimensionality of the feature vector down to 2 dimensions. The aim of t-sne is to place samples with large distance in the high dimensional space in large distance in the lower dimensional space as well. This step not only allows for the visualization of the data in 2 dimensions, but also for clustering the resulting features into 7 clusters using k-means later on. On the left of Figure 5.1, the t-sne reduced 4096 dimensional VGGNet fc2 feature vectors are shown, one vector for each training video — only one frame per video was used for this step. Clearly, there are some clusters visible in the reduced data; on the right in Figure 5.1 the 7 clusters found by k-means are shown in separate colors.

Figure 5.2 shows samples drawn from two different clusters that were obtained as described. They are visually distinguishable and mostly appear to be belonging to different scenarios. While, *e.g.*, in the left cluster there is mostly clear blue skies, in the right conditions are overcast, indicating that the drives that these samples belong to were recorded in different locations or at different times. Note that there are some outliers in each of the clusters, but for the purpose of the task this split is sufficiently accurate.

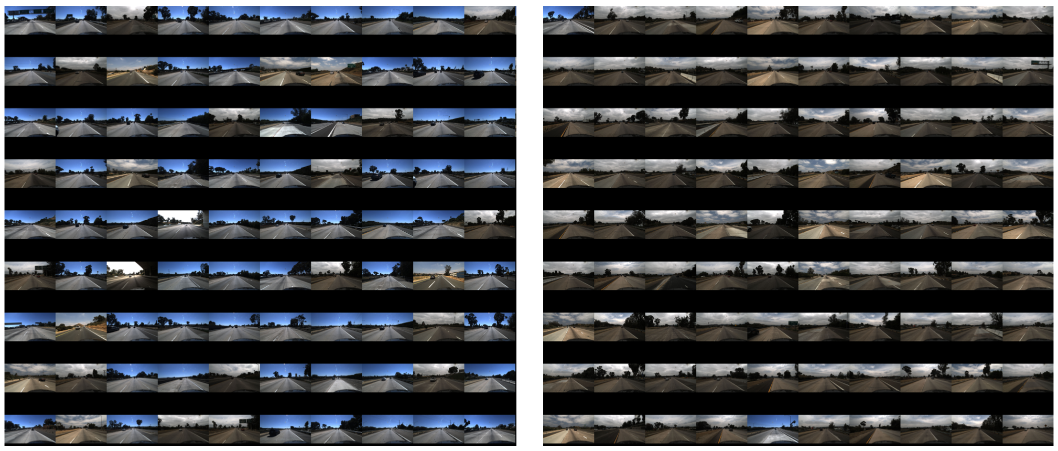

Figure 5.2: Visualization of samples drawn from 2 of the clusters

Finally, to take into account the distribution of different vehicle ranges, one of the clusters consisting of 10% of the total training samples with a (14/63/21%) near, medium, far range distribution is selected as validation set. The remaining data is used for training after defining a fixed 5-fold split for cross validation.

5.2 Evaluation metric

As an evaluation metric for all the experiments, I adopt the evaluation metric that was used for the velocity estimation challenge. For the challenge individual entries are ranked by overall average velocity mean squared error. For each given set of samples C, in this case the ranges near, medium and far, the error is evaluated according to (5.1) and then simply averaged (5.2).

$$E_{V,C} = \frac{1}{|C|} \sum_{c \in C} ||\mathbf{v}_c - \hat{\mathbf{v}}_c||^2 \tag{5.1}$$

$$E_V = \frac{E_{V,near} + E_{V,med} + E_{V,far}}{3}. \tag{5.2}$$

Here, $\mathbf{v}_c$ and $\hat{\mathbf{v}}_c$ denote the ground truth velocity and the estimated velocity of a single vehicle. This metric serves as a solid basis for evaluation, although it is biased towards the smaller partitions C, since if $|C|$, *i.e.* the number of samples in a cluster, is low the average provides less smoothing for extreme values. Hence, in addition to E_V, $E_{V,near}$, $E_{V,med}$ and $E_{V,far}$ are used for evaluation in each experiment.

5.3 Analysis of baseline methods

This section deals with establishing a baseline performance for the task using the methods introduced in Section 4.1.

5.3.1 Challenge baseline

The challenge baseline is the simplest to evaluate, since it consists of all zero input. Evaluating on the validation split derived earlier in Section 5.1 yields the results shown in Table 5.1. Those results serve as the absolute minimum requirement in performance. Any performance worse than this baseline indicates that the method used is basically invalid.

E_V	$E_{V,near}$	$E_{V,med}$	$E_{V,far}$
7.09	3.54	7.04	10.70

Table 5.1: Evaluation results of the challenge baseline on the custom validation split. Errors are given in $\frac{m^2}{s^2}$.

5.3.2 Distance from width baseline

For this baseline approach, which is based on geometry (see Section 4.1.2), bounding boxes are required as input. Both methods of bounding box tracking (see Sections 4.2.1 and 4.2.2) shall be investigated in relation to this baseline approach.

First, the parameters required for this method need to be established, which are the frameskip *i.e.* how far apart frames for the estimation are chosen and the assumed vehicle width to be used. The values for these parameters are chosen by sweeping through both of them simultaneously and obtaining a velocity estimation evaluation result for each combination. For the frame skip, values in the range [1, 39] frames and for the assumed average vehicle width w values within [1.0, 2.2] m in 0.01 m increments are observed. The data for the estimation consists of the combination of training and test data, while leaving out the validation split for evaluation of the method with the parameters thus derived.

Varying frameskip k

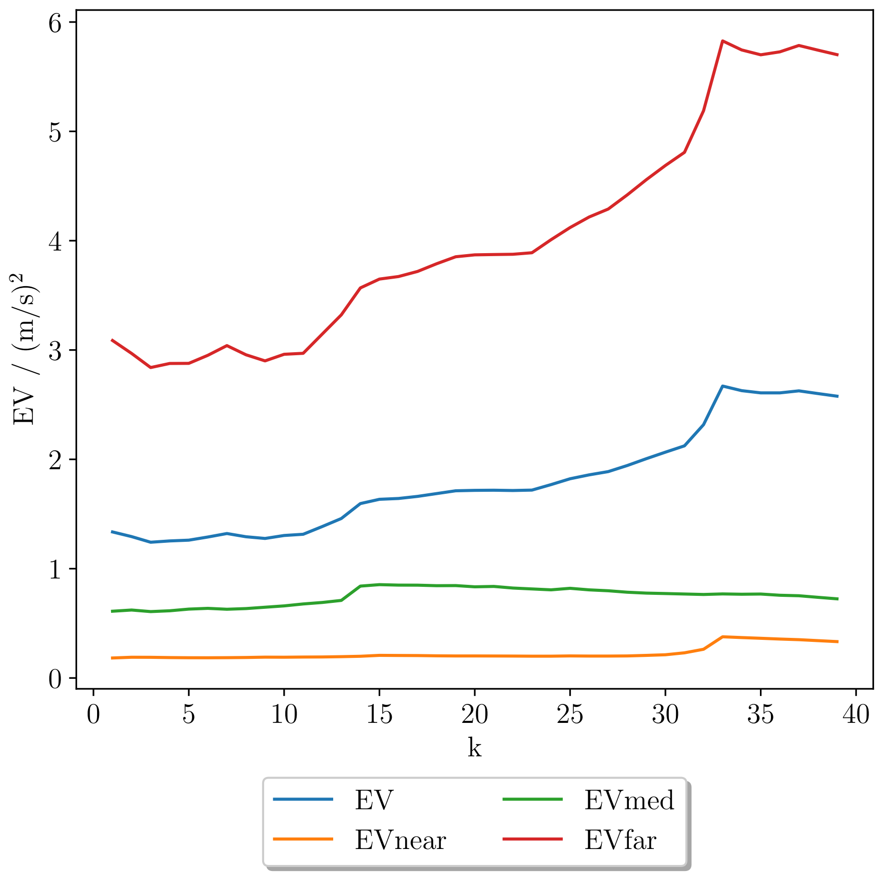

Figure 5.3: Velocity errors over frames skipped for distance from width method using tracked bounding boxes.

The first parameter to be evaluated is the number of frames skipped k in between frames used for estimation. Figure 5.3 shows the evolution of the velocity MSEs for the individual ranges over varying k, when using tracked bounding boxes as features. Here apparently a smaller skip of frames is beneficial, since there is better smoothing provided by the median filter (see Section 4.1.2 for details). Qualitatively, there is little difference between the errors for the different ranges. The minimum for E_V lies at $k = 3$.

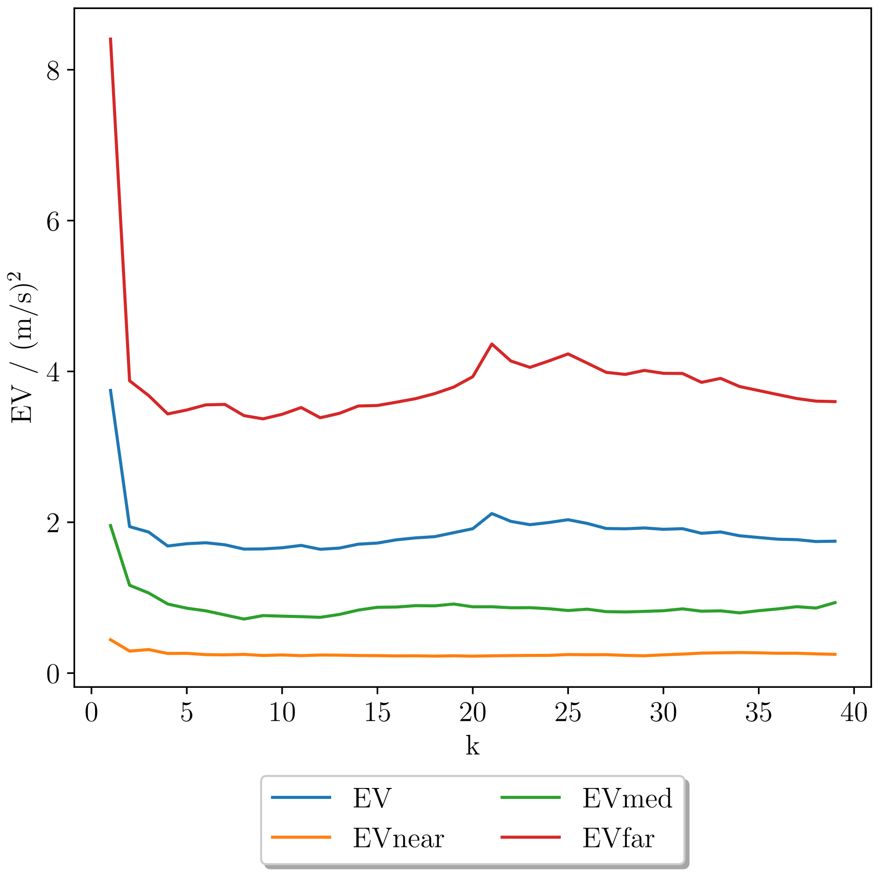

Figure 5.4: Velocity errors over frames skipped for distance from width method using MRCNN.

Figure 5.4 shows the results for the same experiment using MRCNN bounding boxes. The minimum for E_V lies at $k = 12$. Here, interestingly, the errors increase for small values of k, which can be explained by the jitter in the MRCNN tracks. Using

Medianflow and MIL trackers generally yields smoother trajectories. This noise in between frames introduced by the object detection stage makes estimation more stable for larger values of k.

Varying assumed vehicle width w

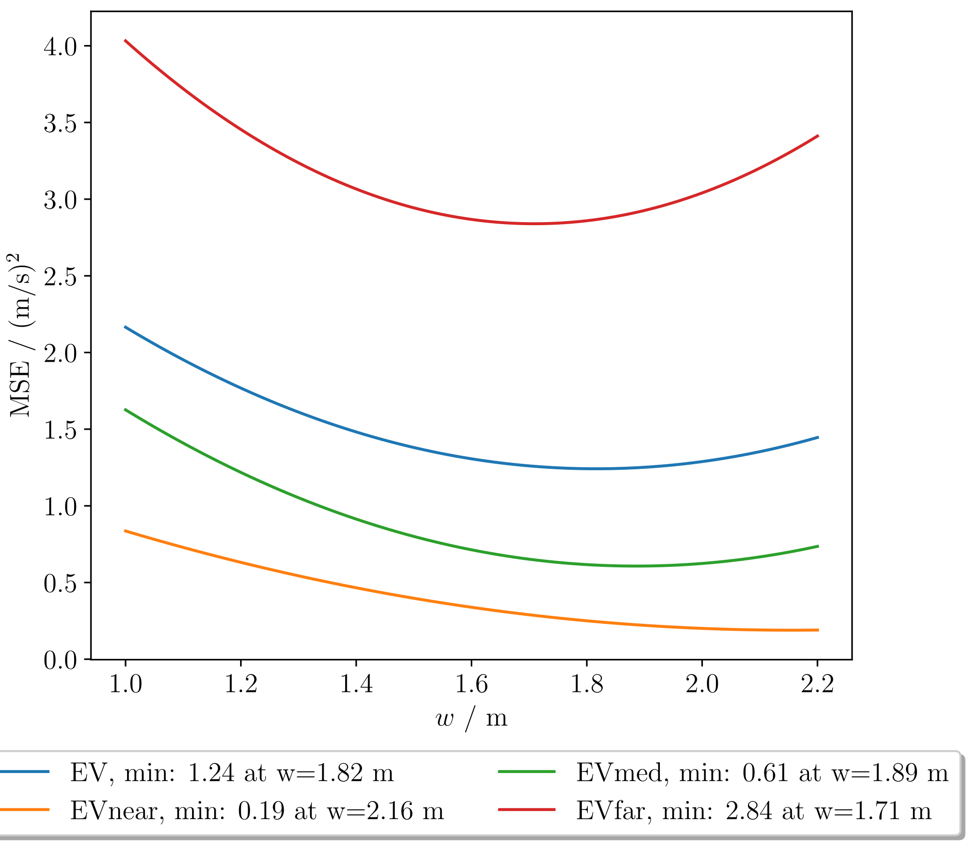

Figure 5.5: Velocity errors over assumed vehicle width for distance from width method using tracked bounding boxes; k = 3.

Next, the evolution of velocity MSEs is evaluated with fixed k for varying w; again for both modes of bounding box extraction. k is chosen such that it corresponds to the minimum of E_V evaluated earlier. The evolution of the velocity MSEs for the individual ranges over varying w using tracked bounding boxes are shown in Figure 5.5. The overall minimum for E_V lies at $w = 1.82$ m, which is valid considering the average vehicle's width. Notably the minimum for close range is at a larger w of 2.16 m, which is due to the shift in perspective for closer range vehicles, where a vehicle that is off to the side is not only seen from the rear, but also from the side. Since the bounding box includes the whole vehicle, the apparent width of the vehicle thus becomes larger. On the contrary, for far range, a slightly smaller width yields smaller velocity estimation errors. Due to the overall smaller appearance at larger range and pixel level errors in the bounding box tracking a, smaller width is justified.

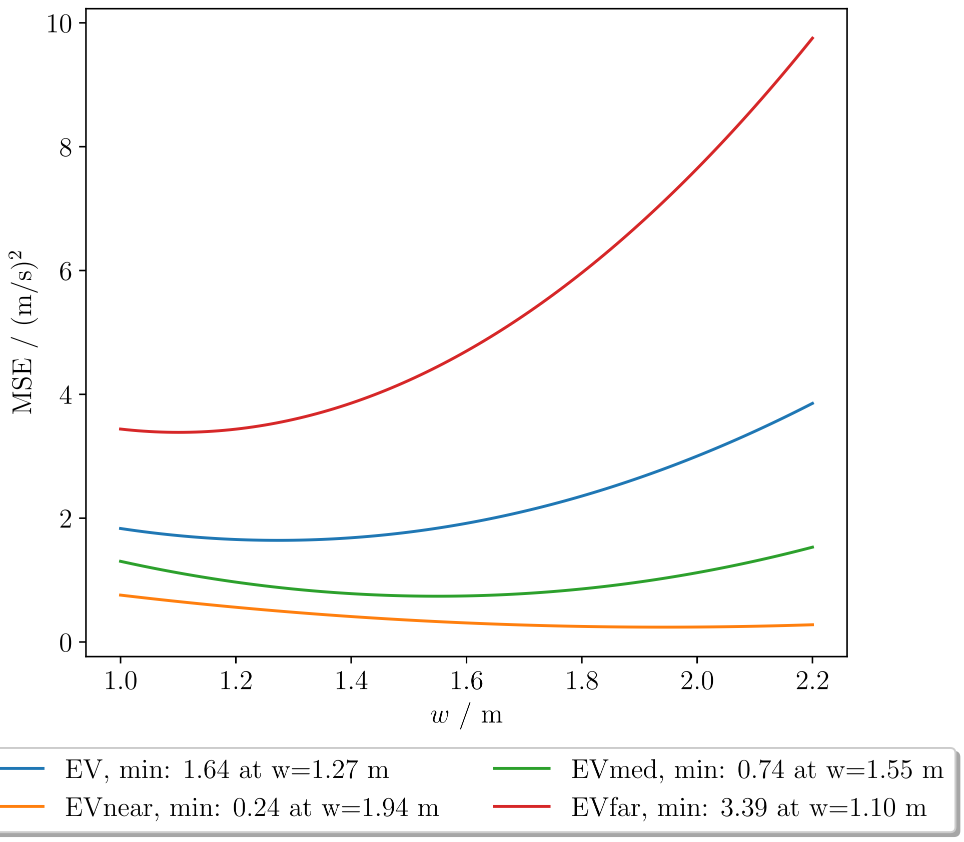

Figure 5.6: Velocity errors over assumed vehicle width for distance from width method using MR-CNN; k = 12.

For bounding boxes obtained with MRCNN (see Figure 5.6), the minimum of E_V lies at $w = 1.27$ m, and also the minima for the other ranges are at significantly smaller widths, when compared to using tracked bounding boxes. This behavior most likely has its roots in the considerably tighter bounding boxes that result from MRCNN. Figure 5.7 shows an example from the training data with bounding boxes for both MRCNN (green) and tracked bounding boxes (red). Besides, for both methods a considerable error in assumed width compared to actual width is inevitable. Considering *e.g.* a distance of 20 m, a difference of just one pixel results in a displacement of 2.8 cm. Assuming that the bounding boxes are true to within a pixel, which would be optimal performance, the estimation of position is still limited by pixel resolution, introducing an error of 5.4 cm. This error further increases to 14 cm at a distance of 50 m.

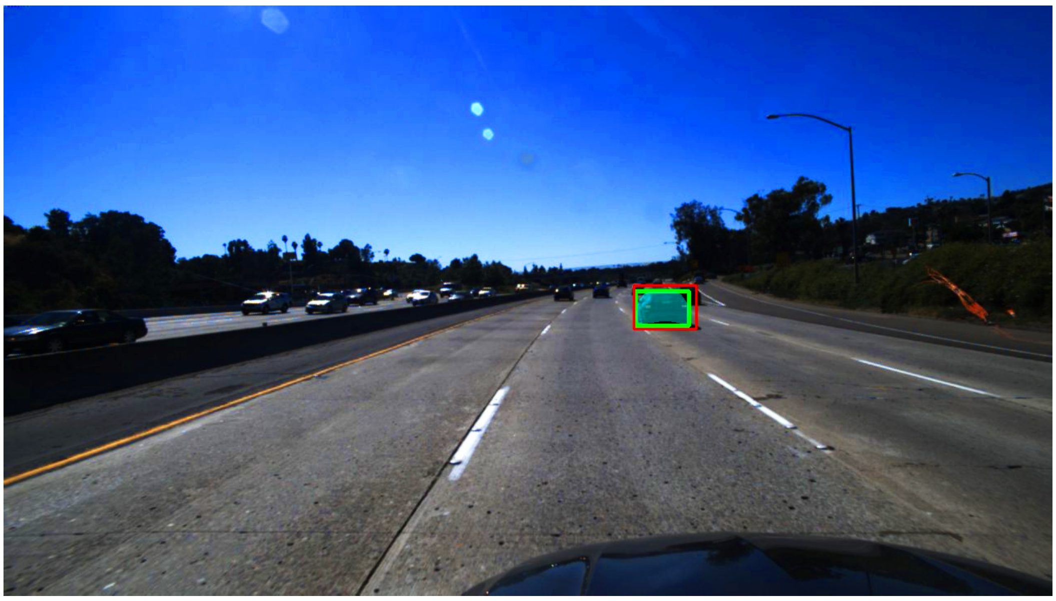
Figure 5.7: Sample from training data showing MRCNN bounding box (green), MRCNN mask (blue) and tracked bounding box (red).

Results on the validation set

The parameters derived earlier for both bounding box generation methods are summed up in Table 5.2.

bounding box method	k	w
tracked bounding boxes	3	1.82 m
MRCNN	12	1.27 m

Table 5.2: Parameters derived for distance from width baseline method.

On the validation set, the distance from width approach achieves the results depicted in Table 5.3. Across all ranges, this baseline outperforms the challenge baseline, although only barely for far range. Far ranges impose the difficulty of little to no variation in bounding box width across the frames, which greatly impairs the estimation performance. The bounding boxes calculated by the MRCNN approach provide a solid base feature for estimation in close ranges, where regular tracking is less accurate due to changes in appearance, perspective and also size. In medium range, the jitter, characteristic for detection based approaches, gives the tracked bounding boxes the benefit over MRCNN.

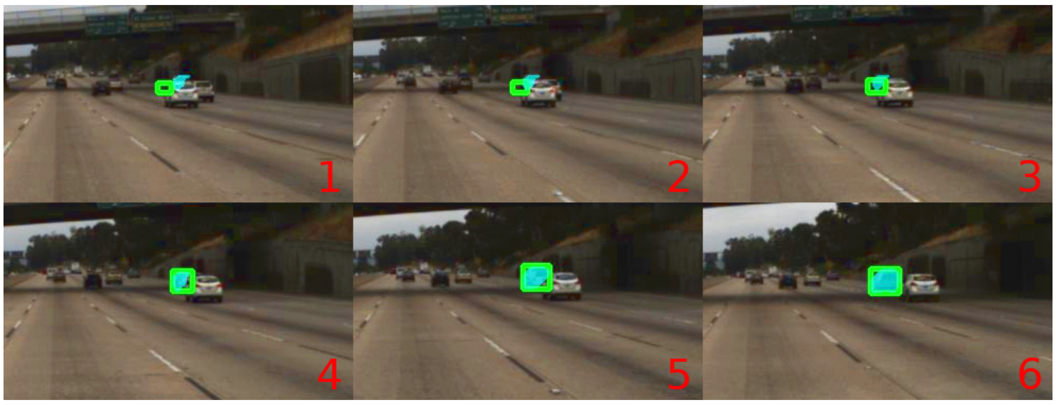

Figure 5.8: Sample of a tracking failure (621 from the training data). Mask (blue) and MRCNN bounding boxes (green) are shown.

The entries in Table 5.3 marked with an asterisk are calculated leaving out one sample from the validation data (sample 621). In this sequence the target vehicle is occluded for part of the sequence, causing a failure in tracking and detection that has a large impact on overall estimation performance of this baseline approach. Figure 5.8 shows some evenly spaced frames from this sequence. Remember that the tracking is running backwards through time, such that the last frame (number 6 in Figure 5.8) is actually the first frame of the sequence seen from the tracker's perspective. About halfway through the sequence, the mask is no longer detected correctly and consequently the IoU is too small, such that the track is substituted with the Medianflow/MIL tracks, which fail in this case due to the occlusion. Leaving this sample out allows a more meaningful assessment of estimation performance in case of stable bounding box tracks. However, this case also highlights that this approach strongly relies on valid inputs, which further implies that it is not robust towards outliers.

bounding box method	E_V	$E_{V,near}$	$E_{V,med}$	$E_{V,far}$
tracked bounding boxes	17.09	1.20	**0.61**	49.45
MRCNN	9.60	**0.63**	1.03	27.15
tracked bounding boxes*	**3.89**	1.20	**0.61**	**9.86**
MRCNN*	**3.89**	**0.63**	1.03	10.00

Table 5.3: Evaluation of the distance from width baseline on the validation set. Entries marked with an asterisk are evaluated with sequence 621 left out.

5.3.3 Distance from height baseline

Analogous to the previous approach, this one too relies on bounding boxes, more specifically the bottom coordinate, denoted p in Figure 4.2. Hence, again MRCNN as well as tracked bounding boxes are considered as base features. While the distance from width approach relies on the assumption of a fixed vehicle width, this approach is free from any such assumptions. Nonetheless, a parameter that has to be set in addition to the frameskip k is the 'virtual horizon' c'_y, *i.e.* the average location of the horizon within the frame. The experiments necessary to choose the parameters are similar to the ones conducted in Section 5.3.2.

Varying frameskip k

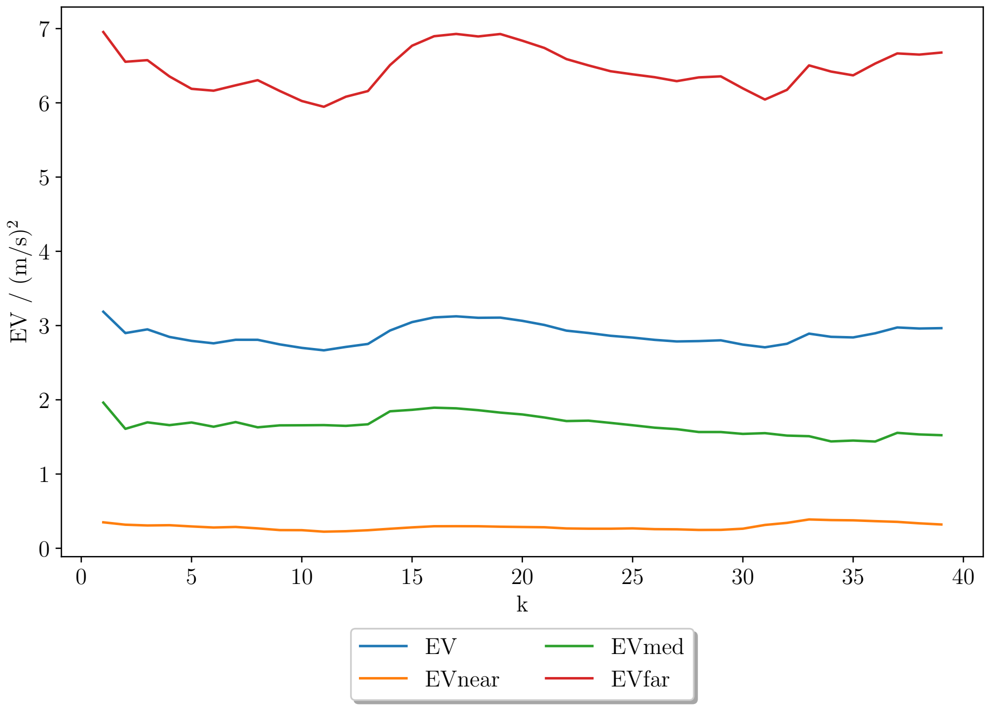

Figure 5.9: Velocity errors over frames skipped for distance from height method using tracked bounding boxes.

Figures 5.9 and 5.10 show the effect that a variation of k has on the velocity MSEs in the individual ranges for tracked as well as MRCNN bounding boxes. Compared to the distance from width method the difference between the two bounding box extraction methods is less pronounced. Qualitative assessment has shown that the bounding box bottom lines for both approaches show very little differences. The bigger deviations lie in the width of the bounding boxes as well as the top border, where especially MRCNN is prone to some noise/jitter over time.

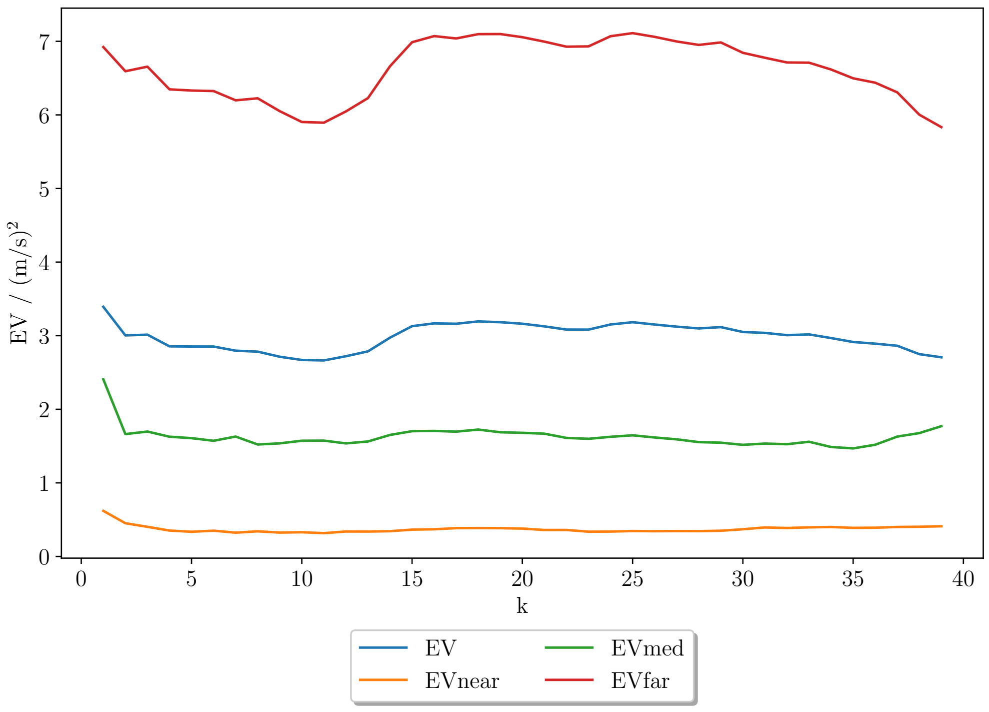

Figure 5.10: Velocity errors over frames skipped for distance from height method using MRCNN.

For both methods, the bottom coordinate of the bounding box is also relatively smooth over time, which results in a quasi constant behavior of E_V over different values of k. The minimum for E_V lies at $k = 11$ for tracked and MRCNN bounding boxes, which highlights the similarity of the two bounding box extraction methods.

Varying assumed 'virtual horizon' c'_y

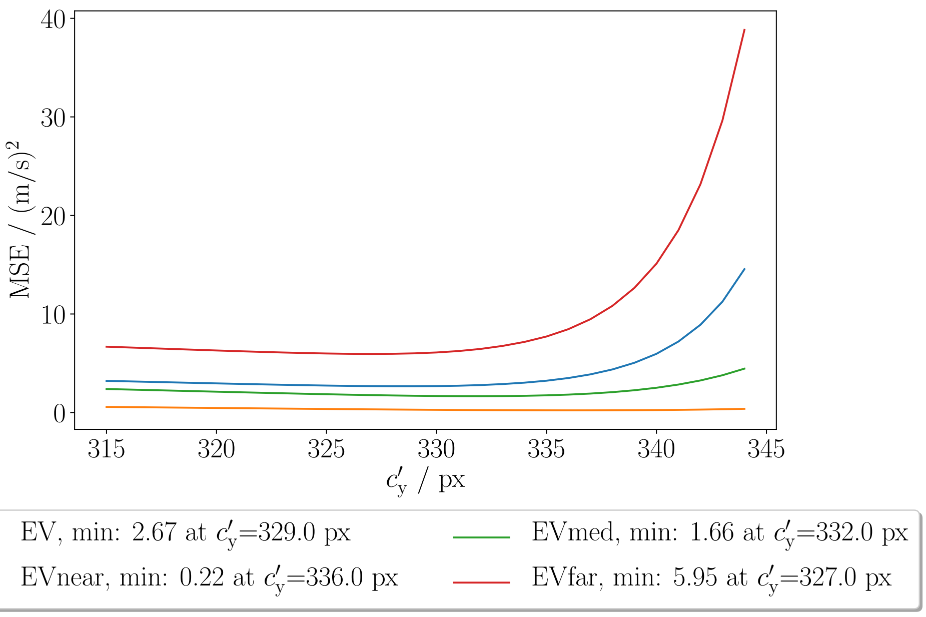

Figure 5.11: Velocity errors over assumed 'virtual horizon' for distance from height method using tracked bounding boxes; $k = 11$.

A variation of the assumed 'virtual horizon' c'_y also impacts velocity estimation performance. It is especially crucial that c'_y is not chosen to be around a critical point. According to (4.5), this critical point is reached any time the bottom bounding box coordinate coincides with the assumed virtual horizon, which evaluates to a distance of $d_x = \infty$. The approach to this critical point is observable in Figures 5.11 and 5.12 for $c'_y > 333$. In the range where $c'_y < 333$ holds there is only a slight increase in MSE towards smaller c'_y. For velocity estimation only relative changes in distance are required, *i.e.* the absolute values of distances are insignificant as long as the relative distances are presserced. Hence, it is important to choose c'_y small enough such that $|p - c'_y| > 0$ holds.

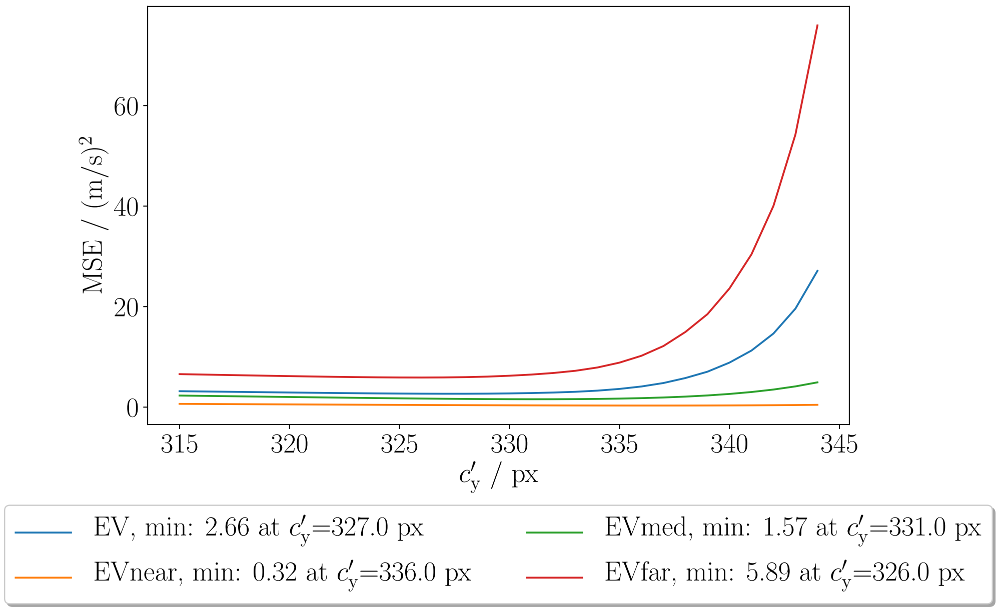

Figure 5.12: Velocity errors over assumed 'virtual horizon' for distance from height method using MRCNN; $k = 11$.

Similar to the variation of k earlier, both MRCNN and tracked bounding boxes yield similar behavior of velocity MSEs over varying c'_y. Since the bounding box bottom borders are most likely close to the horizon for far range samples, those samples dictate the choice of c'_y predominantly. Overall, the influence of the ranges on the choice of c'_y is comparatively small. Still c'_y has to be chosen carefully to avoid divergence of the estimation. E_V reaches a minimum at $c'_y = 329$ px for tracked and $c'_y = 327$ px for MRCNN bounding boxes.

Results on the validation set

The parameters derived earlier for both bounding box generation methods are summed up in Table 5.4.

bounding box method	k	c'_y
tracked bounding boxes	11	329 px
MRCNN	11	327 px

Table 5.4: Parameters derived for distance from height baseline method.

In Table 5.5 the velocity estimation results on the validation set are shown. This approach clearly outperforms the challenge baseline in all ranges, while being on par with the distance from width method in overall performance. It has a slight advantage in far ranges, since the variation in bounding box bottom coordinates is more pronounced than the changes in width. Interestingly it is not performing as strongly as the distance from width approach in medium ranges.

Additionally, the difference between running estimation on all samples versus leaving sample 621 out as before is insignificant, which implies that this approach is more robust towards occlusions, as long as the correct bounding box bottom coordinate p of the tracked vehicle is preserved.

Again, MRCNN is performing well in closer ranges, while struggling in father ranges. On the whole, the difference between the two bounding box extraction methods is not substantial for the distance from height approach.

bounding box method	E_V	$E_{V,near}$	$E_{V,med}$	$E_{V,far}$
tracked bounding boxes	**3.65**	1.32	1.78	**7.84**
MRCNN	3.99	**0.62**	**1.50**	9.83
tracked bounding boxes*	3.66	1.32	1.78	7.87
MRCNN*	3.92	**0.62**	**1.50**	9.63

Table 5.5: Evaluation of the distance from height baseline on the validation set. Entries marked with an asterisk are evaluated with sequence 621 left out.

5.4 Ablation study

Section 4.2 introduced several features extracted from the raw RGB frames provided in the dataset as a basis for vehicle velocity estimation. As submission for the *CVPR'17 Velocity Estimation Challenge*, we originally submitted an approach that combined disparity, optical flow and tracking. To the point of submission, we have not conducted any investigations on the different features and how useful any single feature or combination of several features is for the task. This section is dedicated to shedding light onto this previously unconsidered subject. The pages to follow present an ablation study on the various features and neural network parameters. ReLu, CReLu, ELu, tanh, logistic sigmoid, softplus and softsign activation functions as well as hidden layers with 40 or 70 units are evaluated.

5.4.1 Training

The data split introduced in Section 5.1 serves as a basis for comparison of the different ablations. Since the dataset is relatively small, estimation results can be noisy which results in a large variance in error between training runs. To keep the results statistically relevant, 10 randomly initialized models for each of the 5 partitions and each ablation are trained, resulting in 50 models over which the error is averaged. All of the models trained share the same dropout of 0.2 and the same l_2 regularization of 10^{-5}. Training is stopped early after 500 epochs if the MSE on the validation data does not improve. Also a distinction between training separate models for near, medium and far ranges and training single models for all ranges combined is made.

The following paragraphs first show results for each single feature and later for the best performing combinations of input features.

5.4.2 Tracking

Table 5.6 shows the results obtained when only bounding box tracks (see Section 4.2.1) are used as input for the regression stage. It turns out that tracking comparatively is the single most powerful feature on its own. Apart from near range, training on separated ranges did not prove to be beneficial, which appears valid since for medium and far range there is generally little shift in perspective on the observed vehicle. Thus medium and far range, as well as overall performance is best when training on all of the data, simply because there is more training data available.

In general, the activation function used did not prove to be a crucial choice apart from medium range, where, interestingly, tanh is not a good performer, and networks employing tanh are about 0.2 $\frac{m^2}{s^2}$ higher in $E_{V,med}$. CReLu and softplus perform equally well for medium range.

range	activation	units/hidden layer	E_V	$E_{V,near}$	$E_{V,med}$	$E_{V,far}$
all	tanh	40	**1.86**	1.13	1.00	3.45
near	relu	70	8.83	**0.93**	5.47	20.09
all	softplus	40	3.04	3.12	**0.76**	5.25
all	softsign	40	1.97	1.28	1.20	**3.43**

Table 5.6: Best models found with only tracked bounding boxes as regression input. Models using tracking cues perform surprisingly well across all ranges. Errors are given in $\frac{m^2}{s^2}$, the leftmost column depicts the range of the samples a given approach was trained on. Each row represents the best model in each E_V, $E_{V,near}$, $E_{V,med}$ and $E_{V,far}$.

5.4.3 Motion

It was to be expected that optical flow alone is not enough to provide robust estimates of vehicle velocities. Table 5.7 shows that, apart from near range, optical flow is not very useful. As optical flow estimates depend on pixel level changes, which are generally very subtle for objects that are farther away from the camera, a decrease in performance with increasing range is inevitable. Opposing to the tracking models above, the models based on motion cues benefit more from training on the individual ranges. Interestingly, the best performing overall model is trained on only far range samples. Since the variance in MSE over different runs is rather large for motion based estimation, this can be coincidental. Overall, the estimates based on optical flow are not usable on their own; while they outperform the challenge baseline, they are inferior to other features both in performance and computational cost. Variation of the activation function again did not have a major impact, although a careful choice can give a slight improvement of performance.

range	activation	units/hidden layer	E_V	$E_{V,near}$	$E_{V,med}$	$E_{V,far}$
far	logsig	40	**4.68**	1.95	5.06	7.02
near	softsign	40	4.99	**1.42**	5.41	8.14
med	tanh	70	4.83	2.23	**4.84**	7.42
far	logsig	40	5.06	2.82	5.49	**6.87**

Table 5.7: Best models found with only motion cues as regression input. Estimation based solely on motion cues is inaccurate compared to other features. Errors are given in $\frac{m^2}{s^2}$, the leftmost column depicts the range of the samples a given approach was trained on. Each row represents the best model in each E_V, $E_{V,near}$, $E_{V,med}$ and $E_{V,far}$.

5.4.4 Depth

The results when using only depth as a feature do only slightly deviate from those using only motion cues. As shown in Table 5.8, the only major difference is a decrease in error for near range. Apparently the depth estimation is more accurate in close range, where it provides distances in X with relatively high precision. Additionally, the depth cues seem to be rather consistent across all ranges. With the exception of medium range, training on the specific range alone does not provide any benefit. Compare for example rows 3 and 4 in Table 5.8, where the model in row 3 is trained on medium data only and the model in row 4 is trained on the whole data. The difference in performance between the two models is negligibly small.

range	activation	units/hidden layer	E_V	$E_{V,near}$	$E_{V,med}$	$E_{V,far}$
all	tanh	40	**4.59**	0.97	4.94	7.84
all	logsig	40	4.83	**0.87**	5.31	8.30
med	relu	40	5.61	2.51	**4.88**	9.45
all	crelu	40	4.69	1.38	4.96	**7.72**

Table 5.8: Best models found with only depth cues as regression input. The results are comparable to using motion cues, although better in near range. Errors are given in $\frac{m^2}{s^2}$, the leftmost column depicts the range of the samples a given approach was trained on. Each row represents the best model in each E_V, $E_{V,near}$, $E_{V,med}$ and $E_{V,far}$.

5.4.5 MRCNN tracking

Similar to the other approaches that build on top of convolutional neural networks, MRCNN yields very competitive results for objects that are close to the camera. On the contrary, smaller object scales impose greater difficulties for CNNs, which is also visible across the results for MRCNN based tracking as input feature for the velocity regression in Table 5.9. In close range, the MRCNN tracks perform superior to all other approaches, which is due to the high pixel-level precision that MRCNN elicits especially for large scale objects. In medium ranges, the MRCNN tracks are slightly inferior to the box level tracking, since the MRCNN bounding boxes in this range are subject to more jitter, which in turn impairs the quality of the estimates. In far range, the difference is less pronounced compared to tracked bounding boxes, while the MRCNN features are superior in MSE to all other features. Here, softsign and softplus certainly are the most effective unit activations.

Interestingly, the best performing models for all ranges combined and far range are all trained on medium range samples. I reckon that this behavior stems from the similarity between medium and far range, as well as the high amount of jitter in the bounding boxes for far range. Furthermore, medium range includes considerably more samples than far range, which can provide additional stability for the regressor.

range	activation	units/hidden layer	E_V	$E_{V,near}$	$E_{V,med}$	$E_{V,far}$
med	logsig	40	**1.91**	0.43	1.22	4.07
near	softplus	70	5.05	**0.18**	3.08	11.90
med	softplus	40	3.04	1.66	**1.06**	6.41
med	softsign	40	1.94	0.69	1.38	**3.77**

Table 5.9: Best models found with only MRCNN tracked bounding boxes as regression input. MRCNN tracking is superior to other features in near range. Errors are given in $\frac{m^2}{s^2}$, the leftmost column depicts the range of the samples a given approach was trained on. Each row represents the best model in each E_V, $E_{V,near}$, $E_{V,med}$ and $E_{V,far}$.

5.4.6 Segmentation masks

Another input feature tested are the areas of the segmentation masks provided by MRCNN. The change in area certainly contains similar information as bounding box coordinates; however, no information of the position in the frame is preserved. This definitely limits the ability to estimate lateral velocities, but does not have a large impact on the errors for this dataset since lateral velocities are generally low. Other than in near range, segmentation masks alone do not serve as a sufficient basis for estimating vehicle velocities. The limited performance for farther ranges implies that position in the frame is a beneficial parameter for these ranges, while being less influential in near range. Statistically, vehicles are less likely to exhibit any lateral velocity in closer ranges, since lane changes technically require a safety distance.

Models trained on near and medium ranges perform equally well across all ranges, which might connote that the difference in the temporal evolution of segmentation mask areas between ranges is small and that additionally a relatively simple model is sufficiently accurate in all ranges. This theory is further confirmed by 40 unit models being the best performers across the board, similar to depth as an input feature. Far range segmentation masks are again less useful due to lower SNR across multiple frames caused by inaccuracies of MRCNN.

range	activation	units/hidden layer	E_V	$E_{V,near}$	$E_{V,med}$	$E_{V,far}$
med	logsig	40	**2.71**	0.48	2.39	5.26
near	logsig	40	4.01	**0.23**	3.12	8.67
near	elu	40	3.37	0.31	**2.15**	7.64
med	softsign	40	3.34	2.12	2.99	**4.92**

Table 5.10: Best models found with only MRCNN segmentation masks as regression input. Segmentation masks perform well in near range, but do not outperform any other input feature. Errors are given in $\frac{m^2}{s^2}$, the leftmost column depicts the range of the samples a given approach was trained on. Each row represents the best model in each E_V, $E_{V,near}$, $E_{V,med}$ and $E_{V,far}$.

5.4.7 Best performing methods

The sections thus far have shown how individual input features perform on the task and whether they are of any benefit. This section serves to evaluate the best performing methods including single feature, multiple feature and baseline methods. For the sake of brevity, the three best performing models for each range are summarized in Table 5.11, with the features used indicated.

features used					range	f(x)	units	E_V
track	track*	depth	flow	mask				
all ranges combined								
✓	-	✓	-	-	all	tanh	40	**1.83**
✓	-	-	-	-	all	tanh	40	1.86
✓	-	✓	✓	-	all	tanh	40	1.86
near range								
-	✓	-	-	✓	near	softplus	70	**0.17**
-	✓	-	-	-	near	softplus	70	0.18
-	✓	✓	✓	✓	near	logsig	70	0.22
medium range								
✓	-	-	-	-	width baseline			**0.62**
✓	-	-	-	-	all	softplus	40	0.76
✓	-	✓	-	-	all	crelu	40	0.86
far range								
✓	-	✓	✓	-	all	softsign	40	**3.06**
✓	-	✓	-	-	all	softsign	40	3.23
✓	-		✓	-	all	softsign	40	3.24

Table 5.11: Summary of the best performing approaches. For each separate range the top 3 methods ranked by MSE (E_V) are listed. The first 5 columns from the left indicate the features used in the approach. track refers to tracked bounding boxes and track* to bounding boxes from MRCNN. $[E_V] = \frac{m^2}{s^2}$.

Undoubtedly, tracking is the most useful feature, since it is used in one form or the other in each of the best performing methods (note that track refers to bounding box tracking, and track* to MRCNN tracking). The best performing method over all ranges uses tracking and disparity cues, although the second and third best methods perform just as well. Notably, the second best method relies solely on tracking, which underlines that tracking is an essential feature for the task. The third best approach, which uses all of the features, except for segmentation masks is on par with the second best performance-wise. This is also the full approach that was used for the entry in the *CVPR'17 Velocity Estimation Challenge*. Hence, the ablation study confirms that this is one of the best performing methods, however, a lighter weight and simpler approach would have performed equally well. Intuitively, one would think that combining all of the generally complementary features would yield the best results. I believe that, however, the limited smoothness and consistency of the CNN based features in between frames and the limited accuracy for farther ranges cause a decrease in estimation performance for approaches with all features combined. All of the best performing approaches on all ranges combined are trained on data from all ranges combined, use tanh as activation function and 40 unit layers. A combination of the best models for each range would result in a theoretical $\mathrm{E_V} = 1.28\frac{\mathrm{m}^2}{\mathrm{s}^2}$, which is a significant improvement over the best overall model with $\mathrm{E_V} = 1.83\frac{\mathrm{m}^2}{\mathrm{s}^2}$. Note that the organizers of the *CVPR'17 Velocity Estimation Challenge* communicated a ground truth accuracy of $\mathrm{E_V} = 0.71\frac{\mathrm{m}^2}{\mathrm{s}^2}$ which is achieved using LiDAR, radar and stereo cameras.

In near range, MRCNN is clearly superior to all other approaches. The high pixel level accuracy of MRCNN is its largest benefit, in addition to the robustness to shifts in perspective and appearance, which can be quite dominant in close range. Especially, the other tracking regimes are sometimes too inert to follow the relatively large variation in bounding box size over time. The best performance is achieved when using a combination of the bounding boxes and masks generated by MRCNN, just slightly outperforming an approach using only bounding boxes. Interestingly, the third best approach is the full approach using all features. Apparently, the depth and flow cues cause more confusion than they support a more robust estimate. In contrast, when not using MRCNN, the combination of optical flow and disparities as input features is the best performer at $\mathrm{E_V} = 0.66\frac{\mathrm{m}^2}{\mathrm{s}^2}$. All of the MRCNN based methods are the only ones that benefit from the larger 70 unit layer size and they predominantly employ softplus activations. Additionally, all of the top three near range approaches are trained solely on near range, which makes sense considering the inherent difference to medium and far range examples.

For medium range, the geometry based distance from width baseline method achieves a clear lead over the machine learning based approaches. It relies on tracked bounding boxes which are generally very stable at medium range; while MRCNN bounding boxes might be more accurate, they are more prone to jitter especially at medium and far ranges. The second best approach also relies solely on tracking cues, while the third best also incorporates depth values. Again, the additional depth feature seems to be contrary to the tracking rather than complementary and thus has a negative impact on performance. Both of the machine learning based methods are trained on the whole training data and layers with 40 units work best. They employ softplus as well as crelu activations, which are only mild nonlinearities. The high performance of the baseline approach in medium range results from the linear nature of the estimation problem in this range, where the nonlinearities of the MLP do not provide any better representation. Additionally, the explicit formulation of the baseline benefits from the knowledge of camera calibration parameters, which the implicitly formulated machine learning based approach lacks.

The biggest challenge for far range samples is the small scale of vehicles and thus the limited pixel level accuracy. At father ranges, a single pixel of difference translates to a comparatively large spatial displacement, which can only be countered by a higher resolution sensor. Counterintuitively, the best performing approach for far range employs both depth and flow in addition to tracking and also the second and third placed methods incorporate either depth, flow or both. For most far range samples, the optical flow estimates are rather noisy. In contrast, the depth estimates, while not being pixel level accurate, still provide an approximate information on the relative distance. The best method based solely on tracking achieves $\mathrm{E_V} = 3.43\frac{\mathrm{m}^2}{\mathrm{s}^2}$, which is not too far behind. What it is exactly that the depth and flow cues contribute to the improved estimation capability remains open for further research. Similar to medium range, all far range methods are trained on all of the training data. Additionally, they all use softsign activations with 40 unit hidden layers.

Compared to the best performing baseline approaches the machine learning based approaches achieve an improvement in MSE of 106% for all ranges combined, 135% for near range and 182% for far range. However, in medium range no improvement was achieved, here the distance from width baseline using tracked bounding boxes outperforms all neural network methods by 23%. Furthermore, the most important input feature for all machine learning based approaches is bounding box tracking, which performs reasonably well on its own across all ranges. The tracked bounding boxes are certainly the most consistent of all of the features. While they do not provide low level motion and depth information, they provide relatively smooth and accurate relative changes in both position in the frame and 2D size of the object. In order for the CNN based methods to achieve better performance, an increase in accuracy, consistency and smoothness needs to be achieved.

	Motion	MRCNN	Depth	Tracking	MLP	Baseline
Hardware	GPU	GPU	GPU	CPU	GPU	CPU
Timing	344 ms	229 ms	69 ms	7 ms	3 ms	0.06 ms

Table 5.12: Per frame/vehicle inference time for feature extraction and inference on Intel®Core™ i7-5930K, 32GB, NVIDIA®Titan X/1080Ti.

For any autonomous driving application, real time capabilities on limited computational resources are essential. To properly react to events in traffic, fast computation is required, but due to limited space and available power in vehicles — especially also considering electric vehicles — efficiency with respect to computational resources is crucial. Table 5.12 gives the approximate runtimes for each of the feature extraction stages as well as the inference time using the MLP and baseline methods. Clearly, inference is fast, which is also true for tracking. The feature extraction stages based on CNNs are obviously slower, where monocular depth estimation is the most efficient. MRCNN is rather slow at $\sim$ 4 fps, but comes with the advantage that it provides detection and tracking basically for free. Also a more lightweight MRCNN architecture could be employed; the authors (He et al., 2017) report per image inference times of 117 ms for the `R-50-FPN` approach using a ResNet50 backbone with FPN (see details about MRCNN in Section 4.2.2), which is close to real time. Dense optical flow estimation is the slowest to conduct, although again the most powerful architecture was used. Ilg et al., 2017 report lighter weight architectures with real time capability. From the efficiency perspective, the only CNN based approach that makes sense is MRCNN, if only in near range. For anything else, tracking with an MLP or even simple geometry based estimation method is sufficient and considerably faster. Since the primary goal of this work was to provide an approach with best possible performance w.r.t. velocity MSE, the evaluation of more efficient methods is left for potential future work.

6 Conclusion

This thesis elaborated on the winning approach of the *CVPR'17 Velocity Estimation Challenge* and questioned the efficiency of the original approach. The usefulness of various base features for the task of vehicle velocity estimation was investigated and it was found that bounding box tracks are overall the most useful features. Not only do they serve as a basis for extraction of all other features, but they also allow for accurate velocity estimates when used on their own. Bounding boxes alone preserve sufficient geometric information for the task. Features extracted by convolutional neural networks suffer from limited accuracy at small scales. It turned out that all CNN based methods are only reasonably beneficial to estimation performance in near ranges. Notably, methods based on Mask R-CNN (He et al., 2017) outperformed all other methods by a margin and Mask R-CNN is the only method that is able to detect vehicles in RGB frames, which is a shortcoming of all other methods.

In addition to neural network based methods, methods based solely on tracked bounding boxes and simple geometry were developed. In medium ranges, which include a majority of the annotated vehicles in the dataset, the geometry based methods were not outperformed by any of the neural network based methods. Since tracking comes at comparatively low computational cost, it provides an efficient basis for vehicle velocity estimation in critical real-time applications like autonomous driving scenarios.

Up to the point of this work, the problem of vehicle velocity estimation was not previously tackled, which leaves very limited possibilities for comparison of the developed method to other methods. Nonetheless, the machine learning based method introduced in this thesis has won the *CVPR'17 Velocity Estimation Challenge*, outperforming the second placed approach, which employs a similar strategy based on tracking only, by 15%. Through more careful model selection, a submission post deadline outperformed the second placed approach by 26%; this time also using only tracking. This highlights the importance of tracking features for the task and indicates that further research could focus on improving the vehicle tracks in different ranges.

Unfortunately, no other dataset than the one provided for the *CVPR'17 Velocity Estimation Challenge* readily provides annotations for other vehicle's velocities and position and the acquisition of such ground truth is costly since it requires human annotators. The dataset at hand is rather small, limited to freeway traffic and does not include a large variety of scenarios. Further research would largely benefit from a dataset recorded under numerous different conditions (*e.g.* weather, geographic location) and driving situations (*e.g.* intersections, city traffic), that provides annotations for as many vehicles in the frame as possible.

In the end this thesis proposed an efficient and cheap to implement method for vehicle velocity estimation with close to ground truth accuracy, which is achieved by using a single camera, as opposed to a fusion of stereo cameras, LiDAR and radar. While the method was evaluated on a dataset limited to freeway traffic, an extension to other traffic circumstances is certainly possible and open for future work.

Bibliography

Aufrère, R. et al. (2003). “Perception for collision avoidance and autonomous driving.” In: *Mechatronics* 13.10, pp. 1149–1161 (cit. on p. 2).

Babenko, B., M.-H. Yang, and S. Belongie (2009). “Visual tracking with online multiple instance learning.” In: *Proc. CVPR*. IEEE, pp. 983–990 (cit. on pp. 9, 17, 28).

Baker, S. et al. (2011). “A database and evaluation methodology for optical flow.” In: *International Journal of Computer Vision* 92.1, pp. 1–31 (cit. on pp. 11, 12, 34).

Bradski, G. (2000). “The OpenCV Library.” In: *Dr. Dobb’s Journal of Software Tools* (cit. on p. 28).

Butler, D. J. et al. (2012). “A naturalistic open source movie for optical flow evaluation.” In: *Proc. ECCV*. Ed. by A. Fitzgibbon et al. (Eds.) Part IV, LNCS 7577. Springer-Verlag, pp. 611–625 (cit. on p. 34).

Coifman, B. et al. (1998). “A real-time computer vision system for vehicle tracking and traffic surveillance.” In: *Transportation Research Part C: Emerging Technologies* 6.4, pp. 271–288 (cit. on p. 2).

Cordts, M. et al. (2016). “The cityscapes dataset for semantic urban scene understanding.” In: *Proc. CVPR*, pp. 3213–3223 (cit. on p. 33).

Dai, J. et al. (2016). “R-fcn: Object detection via region-based fully convolutional networks.” In: *NIPS*, pp. 379–387 (cit. on pp. 2, 15).

Dalal, N. and B. Triggs (2005). “Histograms of oriented gradients for human detection.” In: *Proc. CVPR*. Vol. 1. IEEE, pp. 886–893 (cit. on p. 13).

Deriche, R., P. Kornprobst, and G. Aubert (1995). “Optical-flow estimation while preserving its discontinuities: A variational approach.” In: *Asian Conference on Computer Vision*. Springer, pp. 69–80 (cit. on p. 12).

Dosovitskiy, A., P. Fischer, E. Ilg, P. Häusser, et al. (2015). “FlowNet: Learning Optical Flow with Convolutional Networks.” In: *Proc. ICCV* (cit. on p. 33).

Dosovitskiy, A., P. Fischer, E. Ilg, P. Hausser, et al. (2015). “Flownet: Learning optical flow with convolutional networks.” In: *Proc. CVPR*, pp. 2758–2766 (cit. on p. 12).

Duchi, J., E. Hazan, and Y. Singer (2011). “Adaptive subgradient methods for online learning and stochastic optimization.” In: *JMLR* 12.Jul, pp. 2121–2159 (cit. on p. 17).

Everingham, M. et al. (2012). *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. `http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html` (cit. on p. 14).

Garg, R., G. Carneiro, and I. Reid (2016). “Unsupervised CNN for single view depth estimation: Geometry to the rescue.” In: *Proc. ECCV*. Springer, pp. 740–756 (cit. on p. 11).

Geiger, A., P. Lenz, and R. Urtasun (2012). “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite.” In: *Proc. CVPR* (cit. on pp. 1, 2, 10, 33).

Girshick, R. (2015). “Fast R-CNN.” In: *Proc. ICCV* (cit. on pp. 14, 29).

Girshick, R. et al. (2014). “Rich feature hierarchies for accurate object detection and semantic segmentation.” In: *Proc. CVPR*, pp. 580–587 (cit. on p. 14).

Godard, C., O. Mac Aodha, and G. J. Brostow (2017). “Unsupervised Monocular Depth Estimation with Left-Right Consistency.” In: *Proc. CVPR* (cit. on pp. 3, 11, 32).

Grzywaczewski, A. (2017). *Training AI for Self-Driving Vehicles: the Challenge of Scale*. URL: `https://devblogs.nvidia.com/parallelforall/training-self-driving-vehicles-challenge-scale/` (cit. on p. 3).

Hasirlioglu, S. et al. (2017). “Effects of exhaust gases on laser scanner data quality at low ambient temperatures.” In: *Intelligent Vehicles Symposium (IV), 2017 IEEE*. IEEE, pp. 1708–1713 (cit. on p. 2).

He, K. et al. (2017). “Mask r-cnn.” In: *Proc. CVPR*. IEEE, pp. 2980–2988 (cit. on pp. 14–16, 29, 30, 65, 66).

Held, D., S. Thrun, and S. Savarese (2016). “Learning to track at 100 fps with deep regression networks.” In: *Proc. ECCV*. Springer, pp. 749–765 (cit. on p. 9).

Horn, B. K. and B. G. Schunck (1981). “Determining optical flow.” In: *Artificial intelligence* 17.1-3, pp. 185–203 (cit. on p. 12).

Hsieh, J.-W. et al. (2006). “Automatic traffic surveillance system for vehicle tracking and classification.” In: *IEEE Transactions on Intelligent Transportation Systems* 7.2, pp. 175–187 (cit. on p. 2).

Ilg, E. et al. (July 2017). “FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks.” In: *Proc. CVPR* (cit. on pp. 3, 12, 33, 34, 65).

Jurgen, R. K. (2006). *Adaptive cruise control*. Tech. rep. SAE Technical Paper (cit. on p. 2).

Kalal, Z., K. Mikolajczyk, and J. Matas (2010). “Forward-backward error: Automatic detection of tracking failures.” In: *Proc. ICPR*. IEEE, pp. 2756–2759 (cit. on pp. 9, 17, 28).

Kampelmühler, M., M. Müller, and C. Feichtenhofer (2018). “Camera-based vehicle velocity estimation from monocular video.” In: *Proceedings of the 23rd Computer Vision Winter Workshop* (cit. on p. 4).

Kingma, D. and J. Ba (2014). “Adam: A method for stochastic optimization.” In: *Proc. ICLR* (cit. on pp. 17, 40).

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). “Imagenet classification with deep convolutional neural networks.” In: *NIPS*, pp. 1097–1105 (cit. on p. 14).

Kuznietsov, Y., J. Stückler, and B. Leibe (2017). “Semi-Supervised Deep Learning for Monocular Depth Map Prediction.” In: *Proc. CVPR* (cit. on p. 11).

Li, Y. et al. (2016). “Fully convolutional instance-aware semantic segmentation.” In: *Proc. CVPR* (cit. on p. 2).

Lin, T.-Y., P. Dollár, et al. (2017). “Feature pyramid networks for object detection.” In: *Proc. CVPR*. Vol. 1. 2, p. 4 (cit. on p. 30).

Lin, T.-Y., M. Maire, et al. (2014). “Microsoft coco: Common objects in context.” In: *Proc. ECCV*. Springer, pp. 740–755 (cit. on pp. 16, 29, 30).

Liu, W. et al. (2016). “Ssd: Single shot multibox detector.” In: *Proc. ECCV*. Springer, pp. 21–37 (cit. on p. 14).

Lowe, D. G. (2004). “Distinctive image features from scale-invariant keypoints.” In: *IJCV* 60.2, pp. 91–110 (cit. on p. 13).

Lucas, B. D. and T. Kanade (1981). “An Iterative Image Registration Technique with an Application to Stereo Vision (DARPA).” In: *Proceedings of the 1981 DARPA Image Understanding Workshop*, pp. 121–130 (cit. on p. 9).

Lukežič, A. et al. (2017). “FCLT-A Fully-Correlational Long-Term Tracker.” In: *arXiv preprint arXiv:1711.09594* (cit. on p. 9).

Ma, C. et al. (2015). “Hierarchical convolutional features for visual tracking.” In: *Proc. CVPR*, pp. 3074–3082 (cit. on p. 9).

Maaten, L. v. d. and G. Hinton (2008). “Visualizing data using t-SNE.” In: *JMLR* 9.Nov, pp. 2579–2605 (cit. on p. 42).

Mayer, N. et al. (2016). “A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation.” In: *Proc. CVPR*, pp. 4040–4048 (cit. on p. 33).

Menze, M. and A. Geiger (June 2015). “Object Scene Flow for Autonomous Vehicles.” In: *Proc. CVPR* (cit. on p. 2).

Rasshofer, R., M. Spies, and H. Spies (2011). "Influences of weather phenomena on automotive laser radar systems." In: *Advances in Radio Science* 9.B. 2, pp. 49–60 (cit. on p. 2).

Redmon, J. et al. (2016). "You only look once: Unified, real-time object detection." In: *Proc. CVPR*, pp. 779–788 (cit. on p. 14).

Ren, S. et al. (2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." In: *NIPS* (cit. on pp. 14, 29).

Russakovsky, O. et al. (2015). "ImageNet Large Scale Visual Recognition Challenge." In: *IJCV* 115.3, pp. 211–252 (cit. on pp. 13, 41).

Sadeghian, A., A. Alahi, and S. Savarese (2017). "Tracking the untrackable: Learning to track multiple cues with long-term dependencies." In: *Proc. ICCV* 4.5, p. 6 (cit. on p. 9).

Saxena, A., S. H. Chung, and A. Y. Ng (2006). "Learning depth from single monocular images." In: *NIPS*, pp. 1161–1168 (cit. on p. 10).

Shang, W. et al. (2016). "Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units." In: *CoRR* abs/1603.05201 (cit. on p. 39).

Simonyan, K. and A. Zisserman (2014). "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (cit. on p. 41).

Srivastava, N. et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting." In: *JMLR* 15.1, pp. 1929–1958 (cit. on p. 40).

Uijlings, J. R. et al. (2013). "Selective search for object recognition." In: *IJCV* 104.2, pp. 154–171 (cit. on p. 14).

Vogel, C., S. Roth, and K. Schindler (2014). "View-consistent 3D scene flow estimation over multiple frames." In: *Proc. ECCV*. Springer, pp. 263–278 (cit. on p. 3).

Vogel, C., K. Schindler, and S. Roth (2013). "Piecewise rigid scene flow." In: *Proc. CVPR*, pp. 1377–1384 (cit. on p. 2).

Xie, J., R. Girshick, and A. Farhadi (2016). "Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks." In: *Proc. ECCV*. Springer, pp. 842–857 (cit. on p. 11).

Xie, S. et al. (2017). "Aggregated residual transformations for deep neural networks." In: *Proc. CVPR*. IEEE, pp. 5987–5995 (cit. on p. 30).

Xu, D. et al. (2017). "Multi-Scale Continuous CRFs as Sequential Deep Networks for Monocular Depth Estimation." In: *Proc. CVPR* (cit. on p. 10).

Yilmaz, A., O. Javed, and M. Shah (2006). “Object tracking: A survey.” In: *Acm computing surveys (CSUR)* 38.4 (cit. on p. 8).

Zhou, T. et al. (2017). “Unsupervised learning of depth and ego-motion from video.” In: *Proc. CVPR* (cit. on pp. 2, 11).

Zhu, X. et al. (2017). “Flow-Guided Feature Aggregation for Video Object Detection.” In: *Proc. ICCV* (cit. on p. 11).