BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

AUTOMATIC SURVEILLANCE CAMERA CALIBRATION BY OBSERVATION OF RIGID OBJECTS

AUTOMATICKÁ KALIBRACE DOHLEDOVÉ KAMERY POZOROVÁNÍM RIGIDNÍCH OBJEKTŮ

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR — **ING. VOJTĚCH BARTL**
AUTOR PRÁCE

SUPERVISOR — **prof. Ing. ADAM HEROUT, Ph.D.**
ŠKOLITEL

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY

Abstract

Faculty of Information Technology

Department of Computer Graphics and Multimedia

Doctor of Philosophy

Automatic Surveillance Camera Calibration by Observation of Rigid Objects

by Ing. Vojtěch Bartl

This work is focused on automatic camera calibration based on multiple observations of arbitrary rigid objects. Based on observations of rigid objects moving in a common plane, we are able to calibrate camera w.r.t. the plane, and thus we are able to do measurements in a scene. Objects in the image plane are detected, and classified, and landmarks on these objects are localized. Our motivation was the usage of these methods in traffic scenarios, and thus as our "objects" we consider vehicles. We propose three different methods that are able to compute camera calibration based on these localized landmarks in an image plane with the only limitation — 3D models must be provided, but these can be known to the calibration system as a background. The camera calibration process is then fully automatic, and no more information is needed. Contrary to previous state-of-the-art methods for automatic camera calibration, the proposed methods are able to estimate all camera parameters (including focal length).

We also collected a new dataset *BrnoCarPark*, which contains records of different scenes with detected vehicles and localized landmarks. Ground-truth measurements in scenes are available, and these can be re-computed by computed camera calibration parameters. All the proposed methods outperform the recent state-of-the-art method in an accurate manner. We evaluated our methods on the constructed dataset and also another dataset *BrnoCompSpeed*. We also made experiments on synthetic datasets, which prove the stability and usability of the proposed methods.

Keywords

Automatic Camera Calibration, Rigid Objects, Calibration Dataset, Vehicle Detection, Vehicle Classification, Landmarks Localization, Vehicle Re-Identification, Horizon Estimation, AI City Challenge

Klíčová slova

Automatická Kalibrace Kamery, Rigidní Objekty, Datová Sada pro Kalibraci, Detekce Vozidel, Klasifikace Vozidel, Lokalizace Význačných Bodů, Reidentifikace Vozidel, Odhad Horizontu, AI City Challenge

Abstrakt

V této práci popisuji svoji práci během mého doktorského studia. Hlavním výstupem jsou tři různé metody pro automatickou kalibraci kamery na základě pozorovnání rigidních objektů v obraze ze stacionární dohledové kamery (v mém případě vozidel — na syntetických datech bylo ověřeno, že metody fungují s libovolnými rigidními objekty). Objekty pohybující se v určité rovině (např. rovina vozovky) jsou detekovány a na nich jsou lokalizovány význačné body. Při dostupné kalibraci kamery je poté možné provádět měření ve scéně jako např. měření rychlosti nebo rozměrů vozidel. Celý postup je složen z detekce vozidel, jejich klasifikace (určení modelu vozidla) a lokalizace význačných bodů — tyto jednotlivé kroky jsou zajištěny pomocí neuronových sítí.

Při znalosti modelu vozidla je možné použít příslušný 3D model a využít znalostí pozice význačných bodů v prostoru modelu. Tyto pozice v 3D prostoru modelu jsou jediným vstupem algoritmů a mohou být předem známy. Pozice bodů poskytují informace o prostoru a obsahují tak dostatek informací pro následný postup automatické kalibrace kamery. Oproti dosavadním přístupům dokáží navržené metody určit všechny parametry kamery (vnitřní i vnější) včetně ohniskové vzdálenosti — předchozí metody často potřebovaly ohniskovou vzálenost jako vstupní parametr.

Součástí práce bylo rovněž pořízení nového datasetu *BrnoCarPark*, sloužícího k porovnání kvality kalibrace dohledové kamery na základě pozorování vozidel ve scéně. Pro určení kvality kalibrace byla v jednotlivých scénách provedena měření vzdáleností mezi body na vozovce. Při znalosti vzdáleností ve scéně (reálném světě) a pozic bodů v obraze je možné přepočítat vzdálenosti na základě kalibrace a tím zjistit kvalitu kalibrace. Pořízený dataset je veřejně dostupný a slouží dalším výzkumníkům k vývoji jejich vlastních řešení.

Dostupná měření dokazují, že navržené metody překonávají současné metody, které se zabývají podobným problémem. Přestože každá z navržených metod přistupuje k problém trochu jiným způsobem, výsledky všech metod jsou na dostupných datech lepší nežli dosavadní metody. Společným rysem všech metod je detekce význačných bodů na vozidlech a určení modelu vozidla; aby bylo možné použít správné 3D pozice význačných bodů.

Mimo tyto tři navržené metody pro automatickou kalibraci kamery obsahuje práce i informace o mé účasti na různých projektech a výzvách. Během svého studia jsem se zaměřil i na různé výzvy (challenges) pořádané v rámci konferencí. Během těchto výzev jsem řešil různé úkoly a porovnával své výstupy s ostatními výzkumníky z celého světa, což mi pomohlo získat velké množství zkušeností a znalostí. Rovněž práce na různých projektech mi pomohla získat mnoho zkušeností v různých oblastech (strojové učení, počítačové vidění, neuronové sítě, ...).

BARTL, Ing. Vojtěch: *Automatic Surveillance Camera Calibration by Observation of Rigid Objects*. Brno, 2023, PhD thesis, Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing. Adam Herout, Ph.D.

Declaration of Authorship

I, Ing. Vojtěch Bartl, declare that this thesis titled, "Automatic Surveillance Camera Calibration by Observation of Rigid Objects" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Date:

Signed:

Acknowledgements

I would like to thank my supervisor, prof. Ing. Adam Herout, PhD., for his assistance as well as his great and valuable advice throughout all my study. I would also like to thank all my colleagues who participated in my research.

Contents

Chapter 1

Introduction

Camera calibration is an essential task in computer vision, providing information about relations between the 2D image plane and the 3D "real-world". With the knowledge of camera calibration parameters (intrinsic and extrinsic), it is possible to re-compute the position of an arbitrary 2D point in an image to a 3D projection in WCS (world coordinate space). Thus, measurements in the real world can be done. These measurements can be used for different objectives *e.g.* object dimension computation, extended reality, vehicle speed measurement, *etc.* Some more detailed information will be described in Chapter 3.

In this thesis, I want to describe, besides others, three proposed methods for automatic camera calibration [1, 2, 3], which I published. The main goal was to develop a method that should work automatically and should be able to calibrate the camera without the necessity to be physically present by camera or in the scene.

After my master's thesis, "*Mapping the Motion of People by a Stationary Camera*", where I gained some experience with video processing and computer vision, I decided to continue similarly. My first published paper [4] was about horizon estimation by observation of humans/vehicles in the scene. It was partly a follow-up work to the mentioned master's thesis. The first idea was that the located horizon provides information that should later be used for camera calibration (2 vanishing points lie on it). However, it turned out that this would probably not be the ideal direction for future research, and the usage of landmarks seemed to be much more promising.

So, I changed the aim to automatic camera calibration as this was similar to the previous topic, and I considered this topic exciting and valuable. As a first method, I proposed ***LandmarksCalib*** (Section 3.3). As a part of this paper, we collected a new dataset *BrnoCarPark* (Section 3.2.2) for the evaluation of camera calibration in car park scenes. However, due to the prolonged review process in the journal ($\approx$ 19 months), I could not compare the new dataset with the lately proposed methods. It was a little disappointing, and I was sorry I could not use the dataset in later articles I originally had in mind. Finally, the method was published in *Machine Vision and Applications* journal (recent ImpactFactor 3.3, CiteFactor 5.7, *Q2 Quartile*).

During the review process of the first paper, I proposed another method ***OptInOpt*** (Section 3.5). I evaluated this method on *BrnoCompSpeed* (Section 3.2.1) developed by my colleagues earlier, but I could not use *BrnoCarPark* dataset as it was still under review. Not to compare the method to only one method, I created a synthetic dataset (Section 3.2.3) to have some more comparison and the possibility to do some more experiments. This paper was published at *Advanced Video and Signal Based Surveillance* conference (Rank B).

Afterward, I proposed another method for camera calibration. Proposed method ***PlaneCalib*** (Section 3.4) was developed during the review process of the first method. Thus, I still could not use the new *BrnoCarPark* dataset. But the second method *OptInOpt* was already published, and I could compare it with this method. So, the paper compares the *OptInOpt* method and some experiments with synthetic data. This paper was published at *Digital Image Computing: Techniques and Applications* conference (Rank B).

During the development of these methods, I improved the code to run *Differential Evolution* (global optimization method used in my calibration methods as will be described in Chapter 3) in TensorFlow. Thus, the computation was parallel and faster than the standard CPU version. I created a "library" with all my methods and made it publicly available. This "library" is a part of *SmartCarPark* project, which I solved during my study, as will be mentioned later. Word library is in quotes because now it is in the form of source code and not a library in the true sense of the word, but it should be potentially transformed into a library easily.

As all the methods were primarily aimed at calibration by vehicles' observations, I started to cooperate with my colleagues who were working in general on different traffic surveillance tasks — *e.g.* license plate detection and recognition, vehicle re-identification, traffic analysis, *etc.* This brought me to work also on this type of tasks, and I was a co-author of some papers [5, 6] in this area. So, the direction of my further work has partially turned to *Inteligent Transportation Systems* (ITS).

A part of my work was also participation in *AI City Challenge* as a workshop on *Computer Vision and Pattern Recognition (CVPR)* conference. It covers different tasks, and some are often aimed at traffic analysis. We published two papers [7, 8]. The first one [7] covered two Tracks: *City-Scale Multi-Camera Vehicle Tracking* and *City-Scale Multi-Camera Vehicle Re-Identification*. The second one [8] solves the Track called *Multi-Class Multi-Movement Vehicle Counting*. The information about our participation can be found in Section 4.1. Although we did not have great results, it was an interesting experience, and I gained much knowledge.

In 2022, as a part of *AI City Challenge*, a new challenge called *Multi-Class Product Counting & Recognition for Automated Retail Checkout* appeared, and it seemed to be an exciting task. So I proposed a new method to solve this problem — although our solution seemed quite strange [9] (Section 4.2) we placed in the challenge as runner-up with a winning reward (*NVIDIA Jetson Xavier NX dev kit*).

This year, I also participated in a challenge called *Workshop on Maritime Computer Vision (MaCVi)* as a part of *Winter Conference on Applications of Computer Vision*. Here, we tried some approaches to detecting objects in the sea, which can be seen by flying drones above them [10]. We did not reach great results, but it was an exciting experience to work with another type of data than usual. I also got in touch with some recent architectures based on visual transformers such as *DETR* [11] or *Pix2seq* [12].

During my Ph.D. study, I also participated in many projects. Some interesting are, for example:

VRASSEO
Tools and methods for video and image processing to improve effectivity of rescue and security services operations
In cooperation with: Ministry of the Interior of the Czech Republic
During the project's solution, I aimed to develop a system for detecting and tracking persons and vehicles. This system was integrated into an internal solution based on sending messages into queues. I got in touch for example with Faster RCNN [13], YOLOv3 [14], YOLO9000 [15], SORT [16], and DeepSORT [17].

SmarkCarPark
Surveillance Monitoring, Analysis and Re-identification of Traffic for Enhanced Car Parking
Supported by: Technology Agency of the Czech Republic
In cooperation with: FT Technologies a. s. — company
This project was probably the closest to the core of my thesis — automatic camera calibration. The goal was to develop a system for monitoring the situation in car parks. As mentioned, my "library" for camera calibration is a part of project outputs.

VirtualTraining
Research and development of algorithms for processing of videosequences
In cooperation with: VirtualTraining s.r.o. (Rouvy) — company
This was contract research for the company engaged in realistic indoor cycling. The project has many problems to solve — my tasks were, for example, anonymization of humans, detection of obstacles, segmentation of a road, super-resolution, *etc.* . I had to solve different problems, so I worked for example with YOLOv5 [18], SoftTeacher [19], SegFormer [20], PSPNet [21], RetinaFace [22], THOR [23], SiamMask [24], FuseFormer [25], LaMa [26], E2FGVI [27], and SwinIR [28].

NiSiD
Non-Invasive and Secure IDentification
Supported by: Technology Agency of the Czech Republic
In cooperation with: Veracity Protocol s.r.o. — company
The project is aimed at the re-identification of any object. The main idea is to make this process only by taking a photo of the object and the possibility to re-identify this object by taking a new photo any time later. Objects can be very similar (*e.g.* same cloth), but two identities must be recognized. Due to the necessity to capture fine details, we experimented with Transformers [29], Reranking Transformers [30], or CMT [31].

Thanks to different projects, I gained much experience and practice in machine learning, computer vision, neural networks, *etc.* during my Ph.D. study. The list of projects is incomplete, but I consider these the most interesting.

All the proposed calibration methods are compared primarily with ***AutoCalib*** [32] method as it was a state-of-the-art solution at the publication time of the proposed methods.

The most important contributions presented in this thesis are:

- Three different proposed methods for automatic camera calibration based on observations of rigid objects in a scene
- Publicly available implementation of all the proposed methods with TensorFlow optimization speed-up implementation[1]
- Construction of *BrnoCarPark* dataset for evaluation of camera calibration in real-life scenes, which is made publicly available[2]

[1] https://github.com/BUT-GRAPH-at-FIT/Automatic-Camera-Calibration
[2] https://nextcloud.fit.vutbr.cz/s/JXdfk9frbys88Zz

Chapter 2

Existing Methods for Camera Calibration and Other Computer Vision Tasks

In this part, I want to describe the state-of-the-art methods about *camera calibration* 2.1 and other computer vision steps necessary during the proposed camera calibration methods. These include *vehicle detection* 2.2, *fine-grained vehicle classification* 2.3, and *landmarks localization* 2.4. This Chapter is partly based on information from my previous work.

2.1 Camera Calibration

Datasets

A problem with camera calibration in a surveillance manner is a lack of datasets. There are only a few datasets for camera calibration tasks in general. CVGL dataset [33, 34] is made to calibrate the camera based on two images from the same camera somewhere on the street. DeepSportRadar-v1 dataset [35] contains photos of a basketball board from a distance view; this is slightly closer to my task but still different. Thus, only *BrnoCompSpeed* 3.2.1 is usable for my problem, and this is the reason why we collected a new dataset *BrnoCarPark* 3.2.2.

Methods

Typically, camera calibration is made by inserting a suitable pattern of known properties in the camera view — method popularized by Zhang [36] — a planar printed checkerboard is used. Not only planar checkerboard is necessary, but arbitrary planar and non-planar [37, 38] objects can also be used. Although this approach is quite old, it is still used nowadays together with interactive solutions [39, 40]. However, these methods are not applicable in a traffic surveillance manner.

Calibration of surveillance cameras can be done by observation of different objects. Some methods use observation of pedestrians to calibrate cameras. Often are used approaches of connection heads and feet of corresponding persons — these serve to estimate horizon line together with the third vanishing point [41, 42, 43, 44, 45]. The main problem is the scale of the scene; the mean height of a human is often used, which can be very inaccurate as it can differ across the world or potentially in time, leading to poor results.

Existing methods for camera calibration applicable in traffic surveillance are mostly based on manual measurements [46, 47, 48], markers [49, 50, 51, 52, 53], vehicle movement [54, 55, 56, 57], or other principles like optical flow, recognition of cars, or license plates [58, 59, 60].

With the exceptions of methods [56, 57, 32], traffic calibration solutions require known dimensions in the scene (e.g. width of lanes, length of dashed markings, the height of the camera, etc.), and thus they cannot be used in a fully automatic manner. The calibration is often constrained to a limited range of viewpoints, and it supports only the straight motion of vehicles.

Maduro *et al.* [46] assume a known angle of the camera and an available width of the traffic lanes. Marker-based methods either use special markers or horizontal road markings. Cathey and Dailey [49] detect the vanishing point of the lane marking with a known lane width. Grammatikopoulos *et al.* [50] assume a camera with zero roll, and they detect the vanishing point of the road markings. He and Yung [51] calibrate the camera from a pattern formed by dashed line markings on the road. Do *et al.* [52] use an equilateral triangle with known dimensions drawn on the road.

Solutions based on vehicle movement typically detect vanishing points in the direction towards the vehicle motion (first vanishing point) and in directions perpendicular to it (second and third vanishing points). Schoepflin and Dailey [55] use a background model to detect lane boundaries in the activity map. The intersection of lanes is assumed to be the vanishing point of vehicle motion. The second vanishing point is detected from vehicle edges. One known length in the scene is required for complete calibration.

Dubská *et al.* [56] proposed a fully automatic calibration method. It assumes straight movement of cars on the road; this condition is usually met on freeways, but it cannot be considered in parking lots, roundabouts, and similar scenarios. Their method uses a particular form of cascaded Hough Transform [61] to search for vanishing points, and the scene scale is inferred from the mean size of observed vehicles. Sochor *et al.* [57] extended this method by more accurately detecting vanishing points and scale inference. They use fine-grained recognition of vehicles and align the bounding box of a known 3D geometry to the observations in the image. The accuracy of this method is sufficient for speed measurement with a mean error of 1.1 kph.

Bhardwaj *et al.* [32] use an approach somewhat similar to all my proposed methods. Within their method *AutoCalib*, passing vehicles are also observed, and landmarks detected on these vehicles are used. Their camera calibration is based on using a *PnP* (Perspective-n-Point) method for each vehicle separately and averaging the outputs. However, landmarks are detected only for a limited coherent view (roughly from the rear), and the average model for all sedan vehicles is used for determining the 3D positions of landmarks. Since their algorithm is based on the *PnP* method, the focal length f must be given as input. We consider the knowledge of the focal length f as too restrictive since we are looking for fully automatic calibration algorithms. *AutoCalib* is also not accurate enough — the error reported by the authors in their article is 8.98 %.

All the proposed methods are compared with *AutoCalib* [32] method, as this was a state-of-the-art solution to a similar problem when our methods were developed.

After publishing all the methods, Kocur and Ftáčnik [62] developed a solution based on the

localization of vanishing points of vehicles by CNN and diamond space. After localizing vanishing points, they can establish scene geometry. The method needs some measurement in the scene to set the scale properly and thus is more limited to "worldwide" usage than our solutions. The solution is evaluated on *BrnoCarPark* dataset and compared to *LandmarksCalib* — the results are on par.

2.2 Vehicle Detection

Datasets

There are not many datasets focused purely on vehicle detection. One of the datasets focused primarily on traffic is UA-DETRAC [63], also used for tracking and containing recordings of the roads with detections of different types of vehicles. Typically, vehicle detectors are trained on the datasets with wider usage and occurrence of many other classes like COCO [64] or PASCAL VOC [65]. Many classes are available, and "car" is one of them. After extracting proper images containing vehicles, it is possible to train on these data.

Nowadays, more and more datasets designed for autonomous driving are available. These can also be used for training (or fine-tuning) the models because there are available detections (segmentations) of vehicles. Although the view from a dashboard camera differs from the surveillance camera, it can help the model generalize and reach better results. Some of these datasets are for example Cityscapes [66], BDD [67], ApolloCar3D [68], Mapillary Vistas [69], or nuScenes [70].

Methods

Today, it is no longer worth dealing with technologies other than those based on Convolutional Neural Networks (CNN) or generally neural networks; therefore, no other will be mentioned here.

Typically, some backbone (*feature extractor*) is the first part of a detection network. A feature extractor has an image on input (*e.g.* $512 \times 512 \times 3$). It proceeds its layers to reduce information contained in the input image to some features in another dimension (*e.g.* $64 \times 64 \times 1024$). These features are more convenient for neural networks, and they can proceed features further. Features extractor can be almost any available (*e.g.* VGG-16 [71], Inception v2 [72], Inception v3 [73], ResNet-101 [74], MobileNet [75], EfficientNetV2 [76], ResNeXt [77], SqueezeNet [78], *etc.*).

Detectors typically fall into two categories: *one-stage* and *two-stage* detectors.

One-stage detectors One-stage object detectors are a class of object detection models that directly predict object bounding boxes for an image in a one-stage fashion. This means no intermediate task, such as region proposals, must be performed to produce an output. This leads to a simpler and faster model architecture than two-stage detectors. On the other hand, some tasks are for one-stage detectors much more complicated; for a long time, one-stage detectors have not existed for segmentation. Nowadays, this restriction releases as some detectors for segmentation appear (YOLACT [79], YOLOACT++ [80])

One-stage object detectors skip the region proposal stage of two-stage models and run detection directly over a dense sampling of locations. One-stage object detectors are designed to prioritize inference speed. They are extremely fast but may not be as good at recognizing irregularly shaped objects or a group of small objects (due to the grid of probabilities, where small groups can be omitted). One of the most popular one-stage detectors is YOLO [81], which has many further modifications — YOLO9000 [15], YOLOv3 [14], YOLOv4 [82], YOLOv5 [18], YOLOv6 [83], YOLOv7 [84], PP-YOLO [85], YOLOR [86], YOLOX [87] — and is still evolving. Another typical representative of one-stage detectors is SSD (Single Shot Detection) [88], RetinaNet [89], CenterNet [90], or EfficientDet [91].

Two-stage detectors Two-stage object detectors divide the detection task into two stages: extracting regions of interest (RoIs) and then classifying and regressing the RoIs. In the first stage, the algorithm generates proposals in which potential objects can be present. During the second stage, predictions are made based on the generated proposals. These types of detectors are often more accurate than one-stage detectors but often at the cost of losing inference speed.

Two-stage detectors were developed before one-stage detectors, and the leading representative is R-CNN [92]. In comparison with today's architectures, this variant was extremely slow. Evolution in the form Fast R-CNN [93] and Faster R-CNN [13] followed. The advantage of two-stage detectors is the possibility of making more complex computations during the second stage. For example, Mask-RCNN [94] computes segmentation masks of objects during the second stage; or PanopticFPN [95] makes panoptic segmentation of the whole image (combination of instance and semantic segmentation). Two-stage detectors are typically based on FPN (Feature Pyramid Network) [96] — *e.g.* DetectoRS [97].

Transformers In the last few years, a significant expansion of transformers began. Transformers [29], in comparison with previously mentioned methods, typically do not use convolution layers but fully connected and attention layers. It started to be popular thanks to its results in natural language processing (NLP) and visual transformers (ViT [98]) came soon; it transforms an image into small patches and works with it as a sequence (a sentence in NLP) and applies attention layers to it. Although transformer technology reached exciting results in some cases, there are works such as CMT [31] which suppress transformers' good results by using "typical" convolutional layers.

Some examples of transformer-based object detectors are: ViDT [99], DINO [100], DETR [11], *etc.*

2.3 Fine-grained Vehicle Classification

Datsets

Compared with other tasks in my pipeline (camera calibration, vehicle detection in a surveillance camera, landmarks on vehicles detection), a couple of datasets are intended for fine-grained classification. One of them is the work of my colleagues Sochor *et al.* [101] called BoxCars116k. Other datasets usable for this task are The Comprehensive Cars (CompCars) [102], Fine-Grained Vehicle Detection (FGVD) [103], or VehicleID (PKU VehicleID) [104].

Methods

Recently, several methods for fine-grained recognition of various objects were published [105, 106, 107, 108], even for vehicles in particular [109, 110, 111, 112, 113, 114].

The task of fine-grained recognition (classification) benefits from extra image information provided by parts of classified objects. However, it cannot be assumed that the location of such parts is known in advance nor that the location is the same for all objects of the same type. Simon and Rodner [105] proposed a method to deal with this problem (during training & test time) by automatically discovering and locating such parts.

Lin *et al.* [106] and Gao *et al.* [107] approach this problem differently by using *Bilinear Pooling*. Lin *et al.* [106] uses a bilinear classifier [115] to classify features extracted by convolutional layers from a CNN. Gao *et al.* [107] improved this idea and proposed the method for *Compact Bilinear Pooling*, reducing the number of features used while preserving classification accuracy. The method proposed by Lin *et al.* [108] uses three different CNNs for localization, alignment, and classification of images for general object recognition. Sochor *et al.* [114] show that these general fine-grained methods are not accurate enough for vehicle fine-grained recognition, and specialized approaches must be used.

A considerable group of existing fine-grained recognition algorithms are specialized in the classification of vehicles. Some of them are limited to frontal/rear images of vehicles: Pearce and Pears [109] use the detection of license plates to localize the front/rear part of the vehicle for feature extraction, as these parts are usually very discriminative to recognize the vehicles. Directly extracted features from frontal images of cars and exploiting the common structure of the vehicle's frontal mask are presented in the work by Zhang [110]. A more complex method based on optimizing/fitting vehicle 3D CAD model to image data for fine-grained classification was proposed by Lin *et al.* [111].

State-of-the-art results in this field are achieved by methods based on Convolutional Neural Networks. Liu *et al.* [112] proposed to use *Deep Relative Distance* trained on the re-identification task to extract more discriminative feature vectors and *Coupled Clusters Loss* function during training. On the other hand, Sochor *et al.* [114] improve the classification accuracy using an "unwrapped" version of the 3D bounding box of detected vehicles to a 2D plane as an additional input of CNN for fine-grained recognition and other modifications.

Hu *et al.* [116] proceed vehicle's tracking and estimate their 3D orientation. Further 3D orientation estimation is merged with CNN extracted features, and recurrent neural network (RNN) is used to employ visual and temporal information to make classification. As mentioned in vehicle detection, transformers are more often in all parts of computer vision, and fine-grained classification is no exception. For example, He *et al.* [117] says that classical CNN approaches often locate some significant areas and use these localized features to classify images. On the other hand, transformers, thanks to their properties (from NLP), view a much wider area and do not lose information from the whole image, as sometimes can happen to CNNs.

2.4 Landmarks Localization

Datasets

In comparison with fine-grained classification in landmarks localization, there is only one dataset intended concretely for the task of landmark localization on vehicles (or at least I was not able to find any other). Wang *et al.* [118] manually annotated 20 landmarks for the whole VeRi-776 dataset [119]. All other datasets contain some landmarks intended for facial landmarks or human pose estimation (joints). These can be considered somehow similar (in a manner of "landmarks"), but the data are different. An excellent example of a dataset can be COCO-WholeBody [120] containing various types of annotations — hand keypoints, body keypoints, and face keypoints.

Methods

To localize landmarks in the 2D image, we use CNN architecture by Wang *et al.* [118] (OIF, *Orientation Invariant Features*) — proposed for vehicle re-identification. It is based on Stack Hourglass architecture by Newell *et al.* [121] proposed initially for human pose estimation. So, any architecture proposed for human pose estimation should probably be easily modified to locate the vehicles' landmarks. Some standard architectures like HRNet [122] should be fine-tuned to "vehicles", or DWPose [123] based on knowledge distillation should also work very well. As everywhere nowadays, transformers like ViTPose [124] can be potentially used also.

Chapter 3

Proposed Methodology to Automatic Camera Calibration

In this chapter, I describe three proposed methods for automatic camera calibration based on the detection of landmarks on rigid objects (typically vehicles) detected in the view of a static camera. This chapter is a simplification and combination of my main papers [1, 2, 3] for better reading. Large parts of the papers are somehow similar, and thus, I created a reduced version based on these papers and containing the most important information that should be more readable.

All methods are similar in the manner of vehicles detection, fine-grained classification of the vehicles, and detection of landmarks (*keypoints*) of the detected vehicles — thus, they share a similar background, which is described at the beginning of this chapter, together with some information how the methods are evaluated. Further, the methods are described separately because each of them works a bit differently. Finally, all methods are compared to each other, and also variants of combining different methods together are also explored.

3.1 The Common Basis of Proposed Methods

In this part, I describe some information to introduce the issue and also the background of all the proposed methods. As all the methods are essentially similar, I decided to describe the common information here, and thus it is not necessary to read the information later repeatedly.

3.1.1 Camera Calibration

Camera calibration is an important step in the majority of machine vision applications. In various surveillance scenarios, calibration, including scale (to tell the position in world units, like meters, not in image units), is of high importance. With the possibility of making measurements in the scene, the speed of movement in the scene can be computed, and this can be very beneficial to intelligent transportation systems and others — thus *e.g.* the estimation of vehicles speed can be done (security on roads), or some type of 3D reconstructions are possible (*e.g.* surveillance systems). While research works treat camera calibration as a solved problem by showing a checkerboard to the experimental camera, practical applications often

require a calibration procedure that is automatized and suitable for large scene scales. There is a large number of surveillance cameras around the world, and the possibility to calibrate them (in order to be usable in machine vision applications) without the necessity of physical presence is essential. Zhang [36] popularized calibration by inserting a suitable pattern of known properties; in his case, a planar printed checkerboard is used, but arbitrary planar and non-planar [37, 38] objects have been used since.

However, in surveillance of real-world scenes, especially with large numbers of processed cameras, it is extremely inconvenient to calibrate the cameras by showing them markers and by making additional distance measurements in the scene (*e.g.* in the midst of the traffic lanes of a highway).

Despite the fact that other methods for the calibration of surveillance cameras have been proposed — these are discussed in Chapter 2 — all of them either have some constraints, limitations or do not reach satisfactory accuracy.

The goal is to develop fully automatic calibration algorithms for surveillance, providing the internal camera parameters, camera's rotation and translation with regard to the ground plane, and also the scene's scale so that measurements can be done in the world units (*e.g.* meters).

I focus on traffic surveillance; all the methods are described on the traffic surveillance problem, and thus vehicles are used as objects of known properties ("markers"), but the algorithms presented here work with any other suitable rigid objects (this was approved by experiments with synthetic data).

The camera projects every point $\mathbf{x}$ in the world 3D homogeneous coordinates $\mathbf{x} = (x, y, z, 1)^\top$ to its 2D screen image $\mathbf{x}' = (u, v, 1)^\top$:

$$\lambda\, \mathbf{x}' = \mathbf{K}\, [\mathbf{R}|\mathbf{t}]\, \mathbf{x}. \tag{3.1}$$

By calibration, we mean obtaining internal (intrinsic) camera parameters $\mathbf{K}$, the camera rotation matrix $\mathbf{R}$, and its translation vector $\mathbf{t}$ that best model such a projection.

The principal point can be assumed in the center of the image (surveillance cameras generally meet this assumption [55, 125]), the pixels are square, and the skew is not present, therefore $\mathbf{K}$ only contains one unknown parameter — the focal length.

The rotation and translation $[\mathbf{R}|\mathbf{t}]$ have six degrees of freedom. Our approaches assume a planar surface (the road) on which the vehicles are moving. As there is no given central point on the ground plane, the origin of the world coordinate system can be potentially everywhere. Since the first two elements of the translation vector, $\mathbf{t}$ indicate the location of the world origin w.r.t. the principal point, we can set these values arbitrarily — further, zero values are used so that the world origin will be projected in the principal point. The third element of the translation vector $\mathbf{t}$ stands for the height of the camera, and thus it is the only parameter of $\mathbf{t}$ which we will be interested in. So in the end, only 5 parameters are sought after — 3 rotation angles, 1 translation value (height), and focal length. Also, an experiment with distortion was done for *LandmarksCalib* method as is described later in Section 3.3.2.

3.1.2 Landmarks Relations

All the proposed methods are based on detected landmarks in a scene and their known positions in a 3D space — object coordinate system (OCS). Solving camera calibration on these known 3D positions and corresponding 2D positions in image space is the main goal. A similar task is being solved by *PnP* (Perspective-n-Point) methods [126, 127, 128, 129, 130]. However, these algorithms require the knowledge of the focal length f, which is unknown in our task, and it must be optimized together with the rest of the calibration parameters. Even *PnPf* (*PnP* with focal length estimation, [131, 132, 133]) methods exist, which are able to find the focal length.

Moreover, the *PnPf* methods have a high error rate in focal length estimation when there are higher values of noise at the locations of the points, which is quite usual in automatically localized 2D landmarks, as will be mentioned later. In our scenario, we want to use more observations of objects in the ground plane, which means that *PnP*/*PnPf* itself is not feasible as it relates to each object separately, without taking into account the relations between individual objects. In our case, the **relations between the observations of single objects are crucial** and carry the necessary information.

3.1.3 Detection, Classification, and Landmarks

Our calibration methods are based on the detection and classification of vehicles in the video frames. Each video frame is processed by a neural network for detection and localization of vehicles (Faster R-CNN by Ren *et al.* [13], in this case, trained on the COD20k dataset published by Juránek*et al.* [134]). The detected vehicles are classified into fine-grained classes (make & model & submodel & model year) by using previously published vehicle classifier [114] by my colleague. For the most common models, landmarks are localized in vehicles' positions by another neural network by Wang *et al.* [118]. This neural network localizes 20 different landmarks in the image with a vehicle present. Some landmarks can be occluded in the given frame; the network also decides about the landmarks' visibility, and only visible landmarks are further used. Also, not all landmarks are used due to their ambiguity, as will be mentioned later.

It should be noted that although I describe the individual tasks of car detection, recognition, and landmark localization as decoupled, in a production implementation, they could and should be merged into a single neural network predicting for each car its type and locations of the landmarks at the same time, similarly to Mask R-CNN [94] or the Panoptic Feature Pyramid Networks [95], which predict a binary mask for each detection and potentially also landmarks can be predicted.

All solutions are based on 2D-3D correspondences, and thus the landmarks must be as precise as possible. Some of the localized landmarks have an ambiguous 3D position — for example, headlights or fog lamps are hard to localize in the 3D space (not enough precisely). Therefore, not all 20 keypoints are used; only the 12 most usable landmarks (4 wheels, 2 license plates, 2 logos, 4 corners of vehicle top).

3.2 Datasets for Evaluation

To evaluate the proposed methods, we detected and classify vehicles on two datasets. The *BrnoCompSpeed* dataset was collected by my colleagues [135], and the *BrnoCarPark* dataset was collected as a part of my thesis. Our novel *BrnoCarPark* dataset with recordings of parking lots is also publicly available and can serve other research. The most common vehicles for which precise 3D models were available were detected and classified in videos. Also, the positions of keypoints were detected with known 3D positions of these keypoints in 3D OCS (object coordinate system).

The vehicle models for which we have precise 3D models are as follows:

- Hyundai i30 Combi MK2 — (a)
- Hyundai i30 Hatchback MK2 — (b)
- Skoda Fabia Combi MK1 — (c)
- Skoda Fabia Combi MK2 — (d)
- Skoda Fabia Hatchback MK2 — (e)
- Skoda Octavia Combi MK1 — (f)
- Skoda Octavia Combi MK2 — (g)
- Skoda Octavia Sedan MK1 — (h)
- Skoda Octavia Sedan MK2 — (i)
- Volkswagen Golf Combi MK6 — (j)
- Volkswagen Passat Sedan MK6 — (k)

Preview of all models can be seen in Figure 3.1. Only vehicles classified as these for which we have available 3D models are relevant and stored for further processing. Also, not all landmarks are stored (as mentioned in 3.1.3) — the 12 most stable points are exported as a part of the dataset.

3.2.1 BrnoCompSpeed Dataset

Although *BrnoCompSpeed* is a dataset made by my colleagues and is not my work, I use it during the evaluation of the proposed methods, and thus some more information is provided here to compare with other datasets. *BrnoCompSpeed* dataset contains recordings on a highway — examples of sessions can be seen in Figure 3.2.

This dataset was made for the speed measurement of vehicles by a single monocular camera. It contains video recordings of highways captured from the bridge above the highway in a traffic surveillance manner, with ground truth measurements on a road plane and vehicles with ground truth speed. Some more detailed information about the dataset can be found in Table 3.1. However, the dataset only contains straight roads because of its purpose of vehicle speed measurement, and thus the movement of the captured vehicles is limited to one direction only — this is a reason why we collected a new dataset containing the non-straight movement of vehicles.

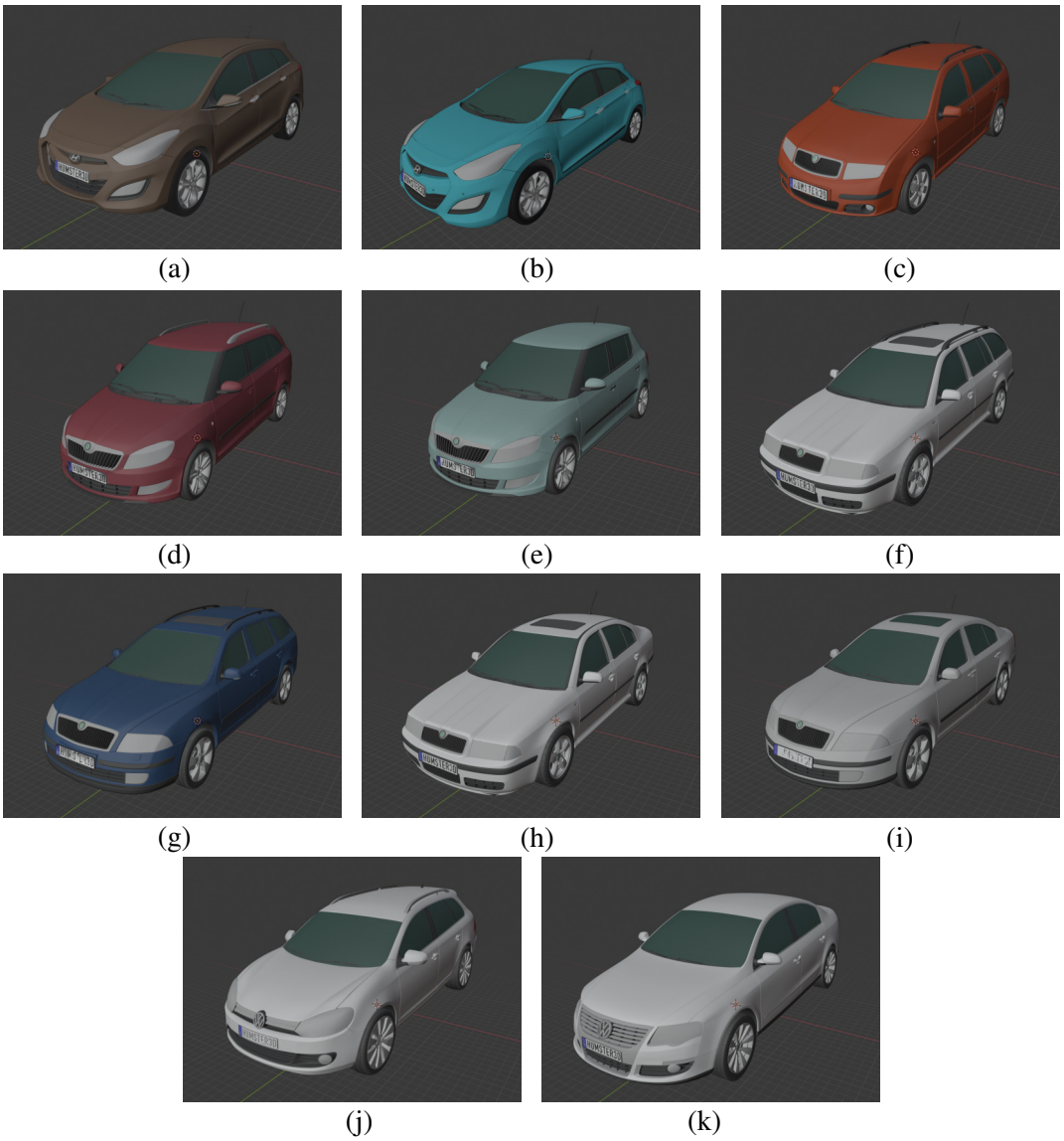

FIGURE 3.1: Available precise 3D models of most common vehicles in datasets. (a) Hyundai i30 Combi MK2; (b) Hyundai i30 Hatchback MK2; (c) Skoda Fabia Combi MK1; (d) Skoda Fabia Combi MK1; (e) Skoda Fabia Hatchback MK2; (f) Skoda Octavia Combi MK1; (g) Skoda Octavia Combi MK2; (h) Skoda Octavia Sedan MK1; (i) Skoda Octavia Sedan MK2; (j) Volkswagen Golf Combi MK6; (k) Volkswagen Passat Sedan MK6.

3.2.2 BrnoCarPark Dataset

In order to be able to better evaluate the proposed methods, a novel challenging calibration dataset *BrnoCarPark* was created. This dataset contains recordings of parking lots with vehicles passing in front of the camera randomly (see Figure 3.3). Therefore, the cars are not moving in any single dominant direction, and the extraction of a single set of vanishing points for the whole scene is impossible. The recordings were captured at two locations from different viewpoints (sessions), during various times of the day and somewhat diverse weather conditions. Some more detailed information about the dataset can be found in Table 3.2.

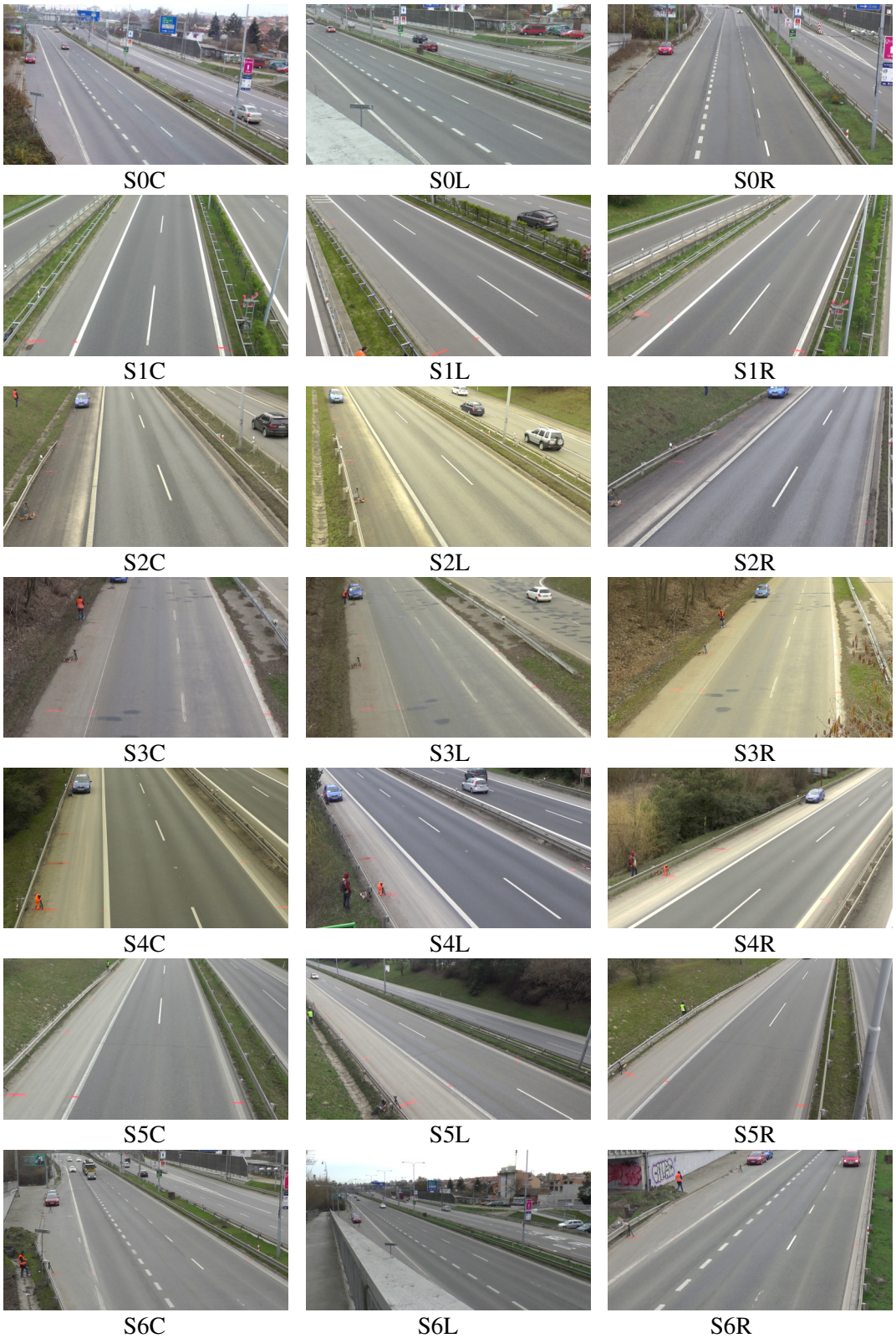

FIGURE 3.2: An example images of *BrnoCompSpeed* dataset.

3.2.3 Synthetic Dataset

Because of problems with the slow review process and for the purpose of experimenting with the influence of noise on landmarks' localization, a synthetic dataset was created. The

				Ground truth	
	Duration	Detections	Keypoints	GT pairs	Mean distance
S0C	01:00:56	6 751	51 831	10	4.9
S0L	01:00:57	10 689	72 869	14	5.5
S0R	00:54:02	13 483	85 206	10	4.9
S1C	01:02:48	3 948	26 991	4	17.88
S1L	01:02:29	6 754	47 041	4	17.88
S1R	01:02:56	6 180	49 028	4	17.88
S2C	00:56:12	7 930	54 905	7	10.26
S2L	00:51:23	9 554	70 166	7	10.26
S2R	01:06:58	10 366	80 636	7	10.26
S3C	00:55:00	667	4 256	10	7.1
S3L	00:55:25	870	6 838	10	7.1
S3R	00:56:14	973	7 102	10	7.1
S4C	01:07:02	7 911	58 092	9	7.88
S4L	01:06:39	11 377	85 224	9	7.88
S4R	01:04:39	9 582	75 882	9	7.88
S5C	01:08:25	9 963	70 587	8	12.48
S5L	01:08:32	19 902	134 634	8	12.48
S5R	01:08:16	11 978	93 929	8	12.48
S6C	00:59:32	31 340	240 137	8	13.11
S6L	00:59:20	27 528	180 312	8	13.11
S6R	01:00:00	21 101	165 486	8	13.11
Mean	01:01:19	10 897	79 102	8.19	9.56

TABLE 3.1: Detailed information about *BrnoCompSpeed* dataset

				Ground truth	
	Duration	Detections	Keypoints	GT pairs	Mean distance
S01	00:47:15	745	4 799	15	18.27
S02	00:47:26	153	1 041	10	24.34
S03	00:46:22	172	1 103	19	16.46
S04	00:46:21	45 283	361 885	17	18.84
S05	00:49:29	740	4 488	16	16.52
S06	00:49:30	36 489	281 418	8	23.02
S07	00:49:04	3 188	22 424	10	16.79
S08	00:49:00	4 520	33 120	12	18.47
S09	15:26:14	2 496	15 412	20	11.05
S10	12:28:01	1 421	9 581	59	14.91
S11	12:28:00	14 353	75 400	50	14.51
Mean value	4:15:09	9 960	73 697	21.45	16.16

TABLE 3.2: Detailed information about *BrnoCarPark* dataset

other benefit is the possibility of verifying the universality of the proposed methods (not only vehicles). The dataset contains passages of various objects (car, cube, block, table, *etc.*) in front of the camera — landmarks similar to those used in real scenarios together with corresponding 3D positions in OCS are generated for each observation. Lanes are generated randomly, and thus not only frontal views are available. Camera parameters are set

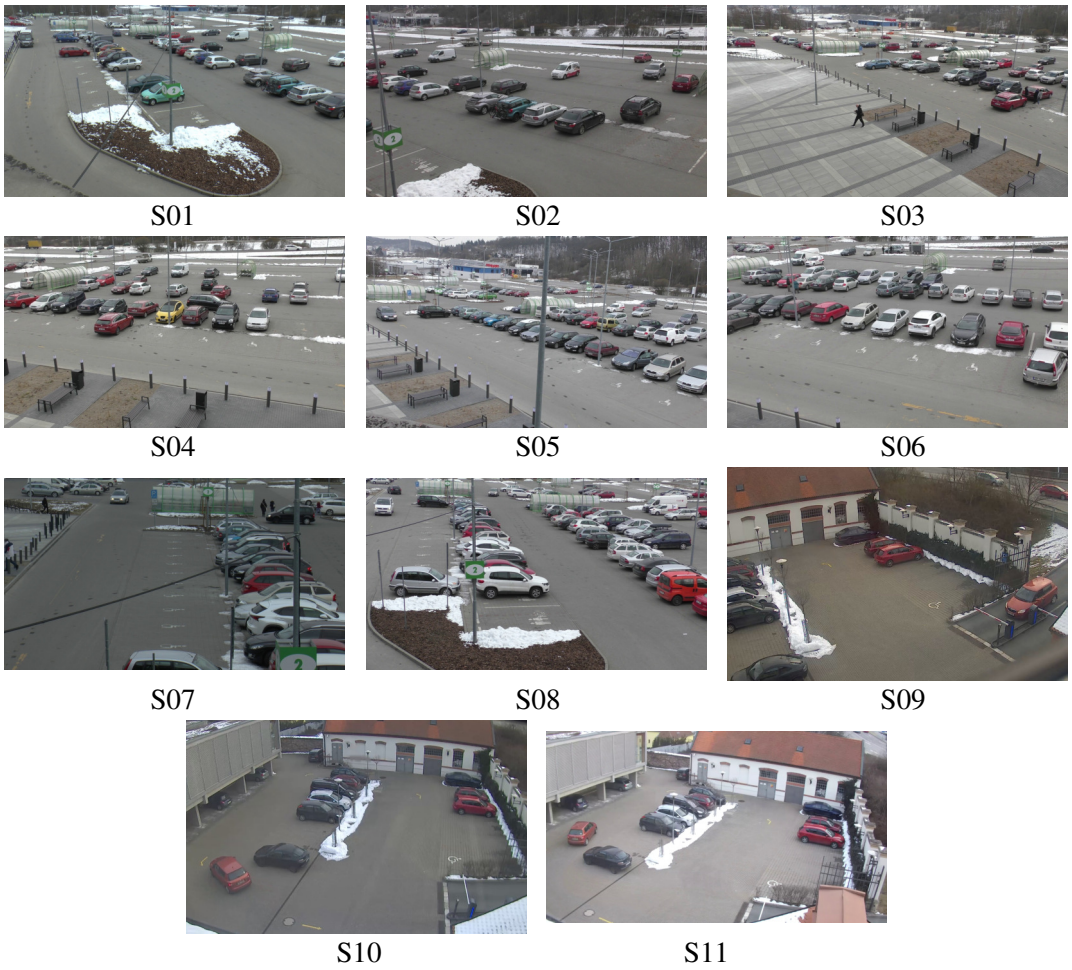

FIGURE 3.3: An example images of *BrnoCarPark* dataset.

randomly in ranges corresponding to the ordinary setting of traffic surveillance cameras. In total, the dataset consists of 40 different scenes, each containing randomly 500 to 4000 objects' passages (93,652 observations in total). Measurements in the ground plane (20 random) and corresponding 2D positions of endpoints are also generated for each scene as the ground truth for calibration accuracy evaluation (3.4.4, 3.5.4). This dataset is also publicly available.

3.2.4 Ground Truth for Evaluation

All the datasets are equipped with ground-truth measurements in the ground plane, which makes the evaluation of the calibration algorithms possible. The ground truth data consists of measurements between various points in the real scene's ground plane (with the exception of synthetic dataset — these are generated) and corresponding 2D positions of the points in the image plane. The existing *BrnoCompSpeed* dataset is equipped with 4 – 10 measurements in each camera view, typically in the direction of the vehicles' movement and in the direction perpendicular to it; the measurements can be seen in Figure 3.4 and mean values of measurements can be found in Table 3.1.

For our new *BrnoCarPark* dataset, we chose a number of distinctive points in the camera images and measured distances between them when the parking lot was empty (at night). We used a laser distance measurer with the precision of $\pm 2\,\mathrm{mm}$ declared by its manufacturer. For

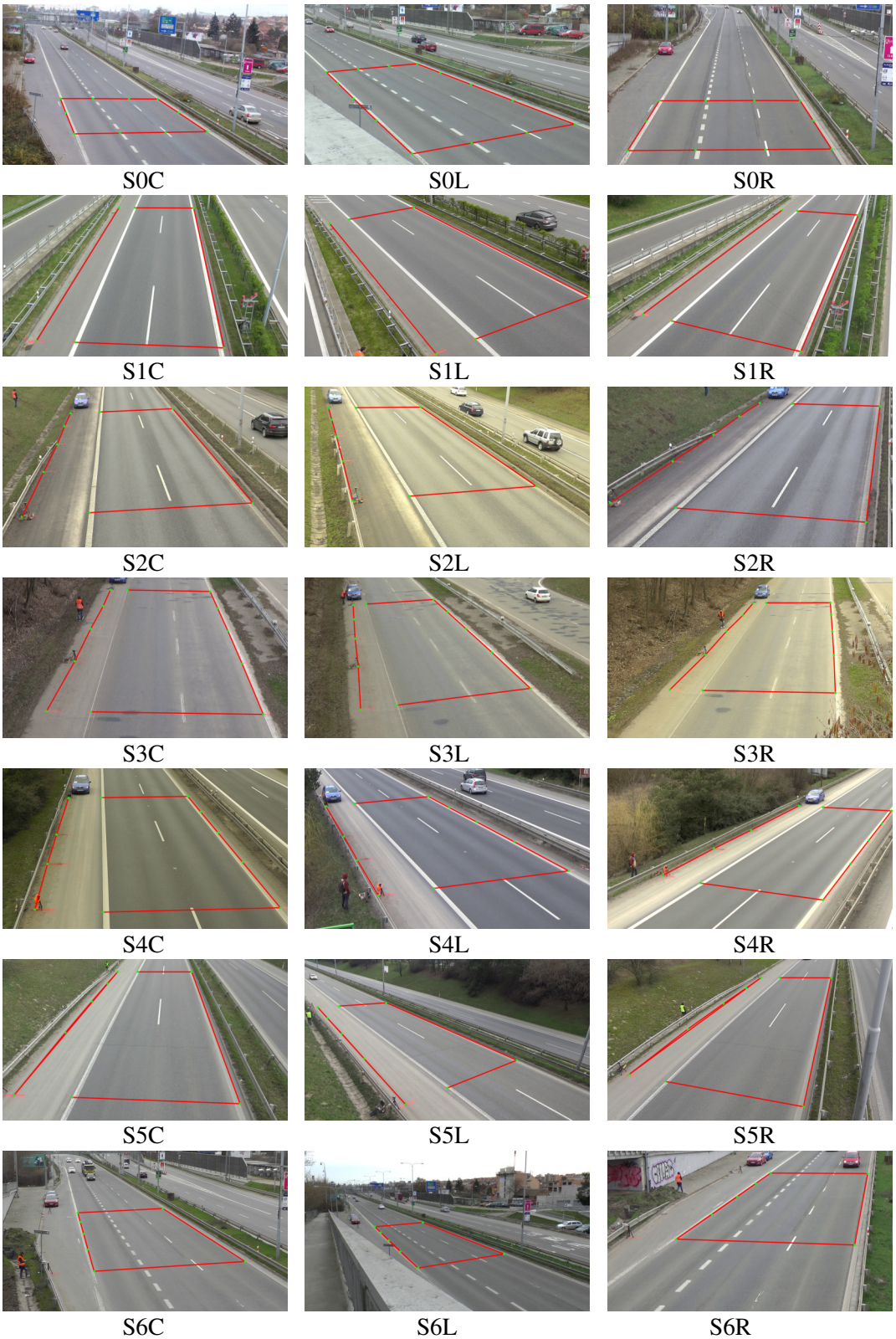

FIGURE 3.4: Groud truth measurements in *BrnoCompSpeed* dataset.

each scene, 8 – 19 distance measurements are available. Ground truth distance measurements with marked 2D points in the frames are depicted in Figure 3.5.

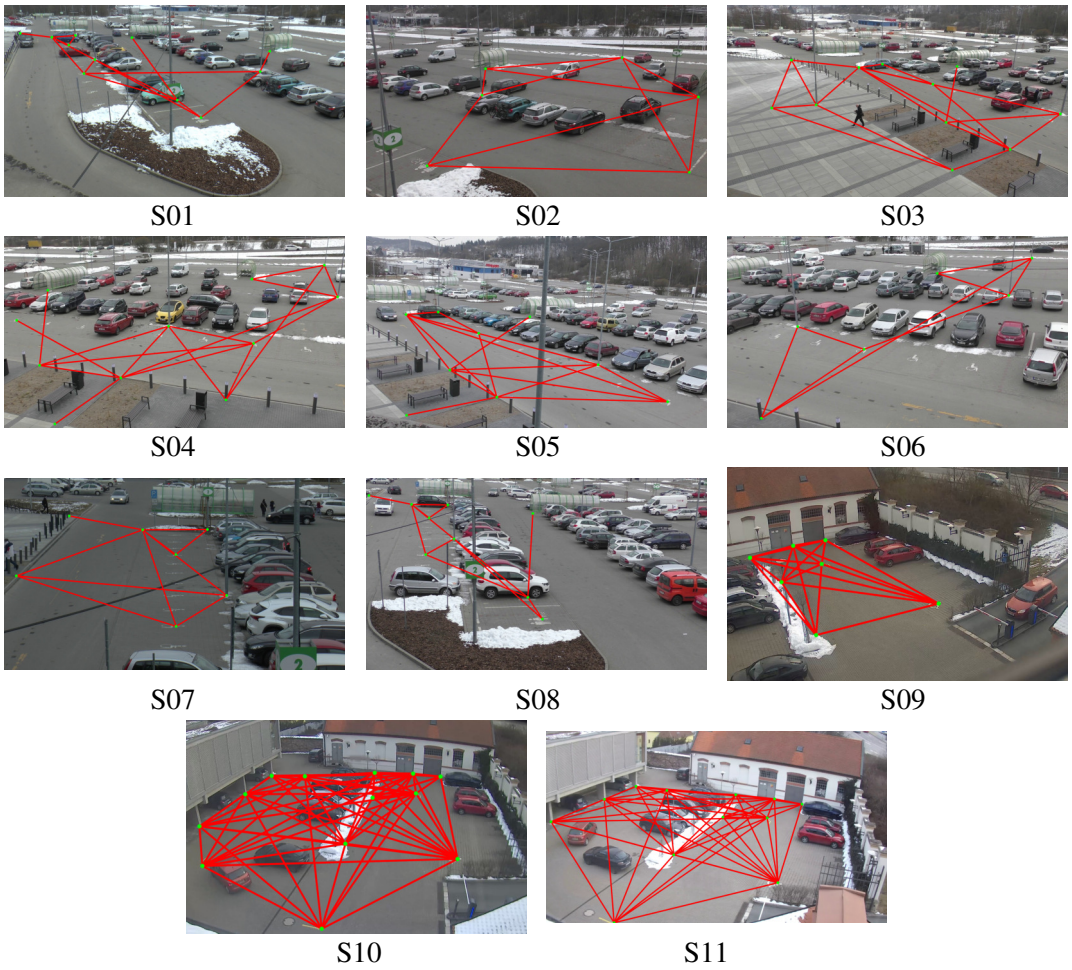

FIGURE 3.5: Ground truth measurements in *BrnoCarPark* dataset.

As mentioned, datasets used for experiments are equipped with ground-truth measurements in the ground plane, which form distances between these points in the real world:

$$\hat{\mathcal{D}} = \left\{\hat{d}_1, \ldots, \hat{d}_D\right\}. \tag{3.2}$$

With the knowledge of calibration parameters (**K**, **R**, **t** — obtained by an arbitrary method), these 2D points can be reconstructed to 3D positions in world coordinates by the usage of the known height of the points above the ground plane, typically 0. Once 3D positions of the 2D ground-truth points are computed, similar measurements corresponding to those in the ground truth $\hat{\mathcal{D}}$ can be computed:

$$\mathcal{D} = \{d_1, \ldots, d_D\} \tag{3.3}$$

and these can be compared against the ground truth $\hat{\mathcal{D}}$ by measuring the relative root mean square error:

$$\text{RMSE} = \sqrt{\frac{1}{D}\sum_{i=1}^{D}\left(\frac{d_i - \hat{d}_i}{\hat{d}_i}\right)^2}. \tag{3.4}$$

This value tells us if the camera is calibrated correctly or if the real-world values that are recomputed by estimated parameters are not correct.

3.2.5 Common Notation of Detections

Evaluation of methods is proposed on datasets composed of videos but processing of these videos is sufficient for easier further steps. The video is transformed into a set of cars' observations:

$$\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_N\} \tag{3.5}$$

and for each car $\mathbf{c}_i$, a set of 2D landmark locations detected by the neural network [118] is available:

$$\bar{\mathcal{K}}^{\mathbf{c}_i} = \left\{\bar{\mathbf{k}}_1^{\mathbf{c}_i}, \ldots, \bar{\mathbf{k}}_K^{\mathbf{c}_i}\right\}. \tag{3.6}$$

Not all detections are used, as we do not have precise 3D CAD models for all vehicles, but only 9 car models are available (details can be seen in Section 3.2). Thus only these vehicles are included in $\mathcal{C}$. These 3D models were manually processed to obtain the accurate 3D positions of the landmarks. Contrary to *AutoCalib* [32], where mean values of these points were used, these locations differ for different recognized vehicle models. For each of the observed vehicles $\mathbf{c}_i$, the correct 3D positions in the vehicle's local coordinate system (OCS) are available:

$$\hat{\mathcal{K}}^{\mathbf{c}_i} = \left\{\hat{\mathbf{k}}_1^{\mathbf{c}_i}, \ldots, \hat{\mathbf{k}}_K^{\mathbf{c}_i}\right\}. \tag{3.7}$$

This notation will be used during a description of all the proposed methods.

3.3 LandmarksCalib: Automatic Camera Calibration by Landmarks on Rigid Objects

In this section, I will describe the method which I consider the most important in my work (and also reaches the best results). It is based on the optimization of calibration parameters based on known distances between keypoint, as will be described further.

3.3.1 Calibration by the Usage of Known Distances

Following notation described in Section 3.2.5 precise 3D positions of the landmarks define the reference 3D distances of the landmarks (pairs of landmarks, identified by indices a and b) as:

$$\hat{\delta}(\mathbf{c}_i, a, b) = \left|\hat{\mathbf{k}}_a^{\mathbf{c}_i}, \hat{\mathbf{k}}_b^{\mathbf{c}_i}\right|, \tag{3.8}$$

which is the Euclidean distance of any two potential landmarks a and b in the model coordinate space. An example of these distances $\hat{\delta}(\mathbf{c}_i, a, b)$, together with a few samples of the detected 2D landmarks $\bar{\mathcal{K}}^{\mathbf{c}_i}$ can be seen in Figure 3.6.

For each landmark's projection $\bar{\mathbf{k}}_j^{\mathbf{c}_i}$, it is possible to compute its 3D position in the world coordinate system based on its known height above the ground plane (the Z-coordinate in the 3D model $\hat{\mathbf{k}}_j^{\mathbf{c}_i}$) and based on given calibration parameters ϕ. This reconstructed 3D position vector will be denoted as $\mathbf{k}_j^{\mathbf{c}_i}(\phi)$, as it is a function of the calibration parameters ϕ, consisting of focal length f (forming the intrinsic matrix $\mathbf{K}_\phi$), rotation matrix $\mathbf{R}_\phi$ and translation vector $\mathbf{t}_\phi$.

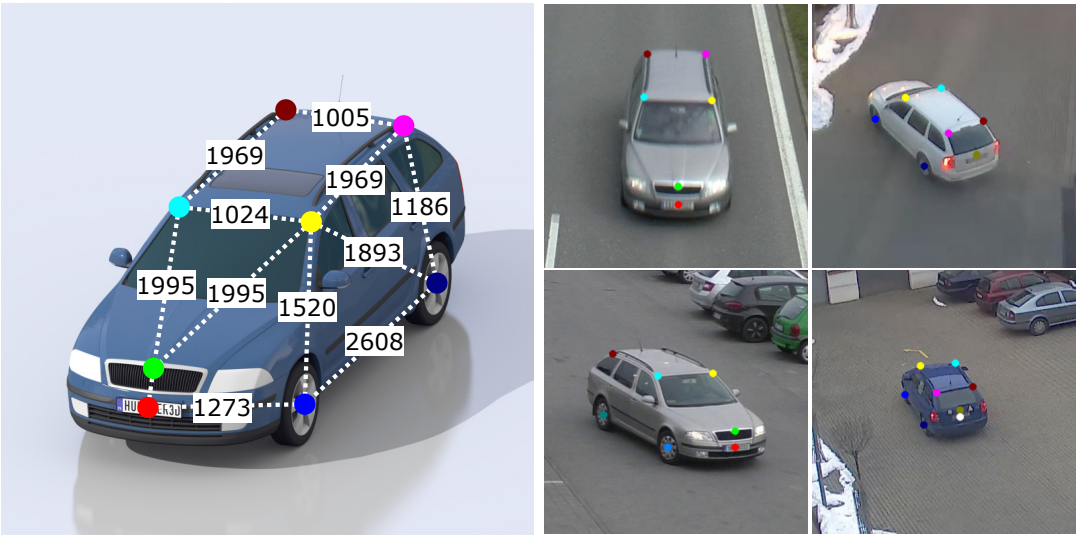

FIGURE 3.6: *left:* Distances $\hat{\delta}$ (in millimeters) in the 3D model of *Skoda Octavia mk2* car. Only a small subset of all the distances in the set $\hat{\mathcal{K}}^{\mathbf{c}_i}$ is shown. *right:* Examples of detected landmarks on actual vehicles of the same type.

Starting with the camera projection (3.1), in our notation:

$$\lambda \begin{bmatrix} \bar{\mathbf{k}}_j^{\mathbf{c}_i} \\ 1 \end{bmatrix} = \mathbf{K}_\phi \left(\mathbf{R}_\phi \mathbf{k}_j^{\mathbf{c}_i}(\phi) + \mathbf{t}_\phi \right), \tag{3.9}$$

which can be rearranged to

$$\mathbf{R}_\phi^{-1} \mathbf{K}_\phi^{-1} \lambda \begin{bmatrix} \bar{\mathbf{k}}_j^{\mathbf{c}_i} \\ 1 \end{bmatrix} = \mathbf{k}_j^{\mathbf{c}_i}(\phi) + \mathbf{R}_\phi^{-1} \mathbf{t}_\phi \tag{3.10}$$

and further:

$$\mathbf{k}_j^{\mathbf{c}_i}(\phi) = \mathbf{R}_\phi^{-1} \left(\mathbf{K}_\phi^{-1} \lambda \begin{bmatrix} \bar{\mathbf{k}}_j^{\mathbf{c}_i} \\ 1 \end{bmatrix} - \mathbf{t}_\phi \right). \tag{3.11}$$

The projective scale λ can be expressed from eq. (3.10) by using the Z coordinate known from the CAD model $\hat{\mathbf{k}}_j^{\mathbf{c}_i}$. Only the third component of all the column vectors is used from (3.10) (operator $[\mathbf{x}]_3$ symbolizes the extraction of the third member):

$$\lambda = \frac{\left[\hat{\mathbf{k}}_j^{\mathbf{c}_i} \right]_3 + \left[\mathbf{R}_\phi^{-1} \mathbf{t}_\phi \right]_3}{\left[\mathbf{R}_\phi^{-1} \mathbf{K}_\phi^{-1} \begin{bmatrix} \bar{\mathbf{k}}_j^{\mathbf{c}_i} \\ 1 \end{bmatrix} \right]_3}. \tag{3.12}$$

For each car $\mathbf{c}_i$ and each pair of landmarks a, b in the world coordinate system $\mathbf{k}_j^{\mathbf{c}_i}(\phi)$, their 3D distance can then be computed as:

$$\delta(\mathbf{c}_i, a, b, \phi) = \left| \mathbf{k}_a^{\mathbf{c}_i}(\phi), \mathbf{k}_b^{\mathbf{c}_i}(\phi) \right|. \tag{3.13}$$

Although localization of the landmarks works well enough (according to [118], 88.8 % of

landmarks are correctly predicted within 3 pixels), it fails significantly in some cases. In particular, it leads to considerable outliers in the detection, which can impact the calibration process significantly. For this reason, we propose to compute the re-projection error for each car $\mathbf{c}_i$ and transform it to weight, controlling the impact of the given sample on the whole calibration. The *PnP* solver [127] provides extrinsic camera parameters for each vehicle instance. Given those, the 3D points from $\hat{\mathcal{K}}^{\mathbf{c}_i}$ are projected to the image plane similarly to eq. (3.9), yielding:

$$\tilde{\mathcal{K}}^{\mathbf{c}_i} = \left\{\tilde{\mathbf{k}}_1^{\mathbf{c}_i}, \ldots, \tilde{\mathbf{k}}_K^{\mathbf{c}_i}\right\}. \tag{3.14}$$

The particular vehicle instance $\mathbf{c}_i$ is then assigned its individual normalized re-projection error:

$$\epsilon(\mathbf{c}_i) = \sqrt{\frac{\sum_{j=1}^{K} \left|\tilde{\mathbf{k}}_j^{\mathbf{c}_i}, \bar{\mathbf{k}}_j^{\mathbf{c}_i}\right|}{\sum_{j=1}^{K} \left|\tilde{\mathbf{k}}_j^{\mathbf{c}_i}, \mathbf{K}^{\mathbf{c}_i}\right|}}, \tag{3.15}$$

where $\mathbf{K}^{\mathbf{c}_i}$ is the mean of all the points $\bar{\mathbf{k}}^{\mathbf{c}_i}$. The fraction normalizes the re-projection error so that vehicle instances of different sizes are mutually comparable. The *PnP* computation assumes knowledge of the intrinsic matrix, which is for now considered to be known, and its estimation is discussed later.

For each vehicle $\mathbf{c}_i$ its normalized re-projection error $\epsilon(\mathbf{c}_i)$ is computed by (3.15), which defines its *weight*:

$$\mathbf{w}^{\mathbf{c}_i} = \left(\frac{1}{\epsilon(\mathbf{c}_i)}\right)^{\alpha}, \tag{3.16}$$

where α controls the power of the weight, and its effect is studied in Section 3.3.2.

The total error/cost of the observations in the video given some calibration parameters ϕ can be expressed as follows:

$$\varepsilon(\phi) = \frac{1}{\mathbf{W}} \sum_{\mathbf{c}_i \in \mathcal{C}} \sum_{a,b} \left(\frac{\delta(\mathbf{c}_i, a, b, \phi) - \hat{\delta}(\mathbf{c}_i, a, b)}{\hat{\delta}(\mathbf{c}_i, a, b)}\right)^2 \mathbf{w}^{\mathbf{c}_i}, \tag{3.17}$$

where $\mathbf{W} = \sum_{\mathbf{c}_i \in \mathcal{C}} \mathbf{w}^{\mathbf{c}_i}$ and thus (3.17) is the weighted mean of vehicles' reconstruction errors. The process of calibration consists of finding such parameters ϕ that minimizes this error function. In our experiments, we find the parameters ϕ by *Differential Evolution* [136] minimizing (3.17). Any other global optimization method can be used; differential evolution was chosen due to its fast computation and robustness. We experimented with local optimizers as well (gradient descent, AdaGrad [137], Adam [138], L–BFGS [139]), but they failed, so the problem appears to be considerably non-linear.

Since available 3D positions of landmarks $\hat{\mathcal{K}}$ are located in the vehicle's (local) coordinate system, projection of these 3D points to the image plane based on the calibration parameters ϕ (by eq. (3.9)) would project all points near the world coordinate origin (3D coordinates are not available in the world coordinate system). However, detected 2D landmarks' positions $\bar{\mathcal{K}}$ are localized in the whole world coordinate system (whole image plane), and thus these projected positions do not correspond to the localized ones. Due to this fact, it is necessary to use the reconstruction error computed from detected 2D landmarks' positions $\bar{\mathcal{K}}$ instead of

the re-projection error in the image plane.

As was mentioned before, weights (3.16) are used in the process of calibration parameters optimization (3.17). However, the computation of weights needs projected points $\tilde{\mathcal{K}}^{\mathbf{c}_i}$ which are computed by a *PnP* solver and thus the knowledge of the focal length is necessary. Since the focal length value is assumed to be unknown, it must be estimated first, and only after it the weights $\mathbf{w}^{\mathbf{c}_i}$ can be computed. Therefore, the whole calibration process is twofold; in the first pass, all weights $\mathbf{w}^{\mathbf{c}_i}$ are set to the value 1.0. The estimated value of focal length in the first pass is used for computing the more accurate weights $\mathbf{w}^{\mathbf{c}_i}$ by eq. (3.15), and these weights are used during the second pass of the calibration process.

The usage of two iterations of computation (and also proper weights) seems to be beneficial since the error is reduced approximately by 53 % as is described in Section 3.3.2.

3.3.2 Results

As explained in Section 3.2.4, RMSE can be computed for ground truth measurements in scene similarly to *AutoCalib* [32], and thus we can compare our results with *AutoCalib* results.

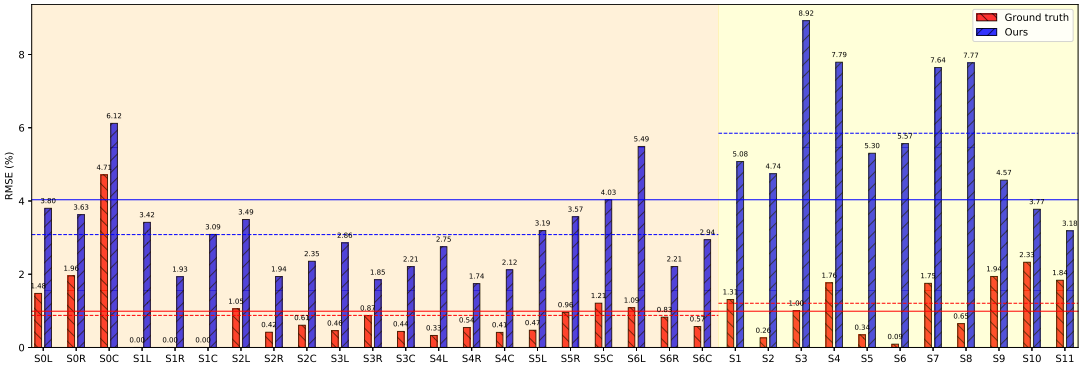

FIGURE 3.7: Accuracy of the proposed calibration method vs ground truth calibration. The red bars describe ground truth calibration, and the blue bars describe proposed algorithm accuracy; red and blue horizontal lines are averages per dataset (dashed) and overall measurements (solid). All is evaluated on *BrnoCompSpeed* (left part) dataset and new *BrnoCarPark* dataset (right part). The ground truth calibration has an average *RMSE* of 0.99 %, and the proposed method has an average *RMSE* of 4.03 %.

Although all the real-world measurements and the corresponding annotations of the 2D points in the scene images were made as precise as possible, some inaccuracies must inevitably occur. In order to quantify these, we made a calibration based on the 2D ground-truth measurements $\hat{\mathcal{D}}$, by using the same methodology as described in Section 3.3.1 (with all the point's Z coordinate being 0). The ground truth calibration errors in the individual scenes are plotted in Figure 3.7 as the red bars. The same graph also shows the error (3.4) of our method for all the scenes (as measured by the more or less accurate ground truth).

We compared our proposed method to *AutoCalib* — the state-of-the-art alternative solution. Since the authors did not make *AutoCalib* code public, we reimplemented their algorithm according to their paper [32]. We used the same landmarks as in our method and also shared the 3D models. We assume that this should constitute an improvement to the original *AutoCalib* method because then only rear views of the vehicles were used, and the set of the landmarks was thus greatly limited. Also, the authors of *AutoCalib* used one united 3D model

representing all sedan vehicles by estimating some average landmark positions, and they do not distinguish between individual vehicle models. Focal length values are necessary for *AutoCalib* method — these are available within *BrnoCompSpeed* dataset; for the *BrnoCarPark* dataset, focal length values computed by ground truth estimation, as was mentioned earlier, were used.

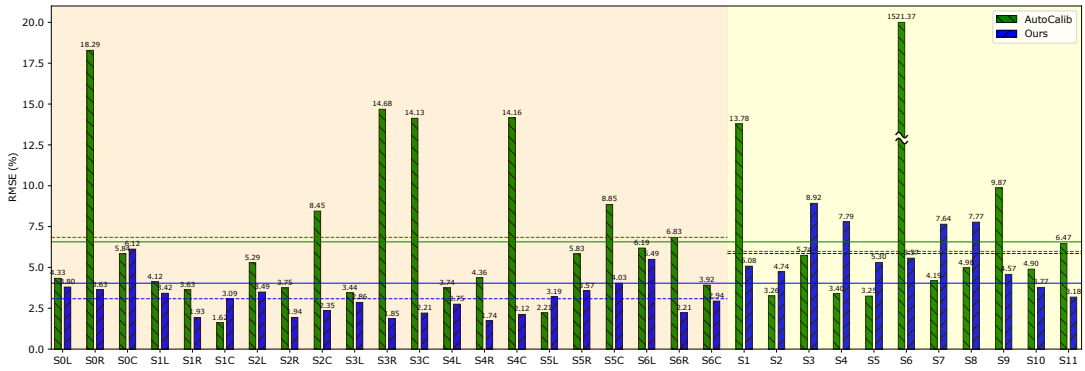

FIGURE 3.8: Comparison of accuracy for the proposed *LandmarksCalib* and *AutoCalib* method. Average results are as follows: *BrnoCompSpeed* dataset – *AutoCalib* 6.84 %, ours 3.08 %; *BrnoCarPark* dataset – *AutoCalib* 5.98 %, ours 5.84 % (it should be mentioned that *AutoCalib* failed significantly on scene *S6* and thus this single scene is not evaluated for *AutoCalib* method); both datasets — *AutoCalib* 6.56 %, ours 4.03 %.

The comparison of our method with *AutoCalib* is shown in Figure 3.8. The mean RMSE across all the scenes was decreased from 6.56 % by *AutoCalib* to 4.03 % by our approach. It should be noted that Bhardwaj *et al.* [32] report the error of 8.98 % on their data in their paper; our implementation thus seems on par or even slightly better than the original solution (though it should be noted that the evaluation dataset is different).

In all our experiments, the *Differential Evolution* parameters are set as follows: population size (number of parents, NP) — 15 times the number of parameters (75); crossover probability (CR) — 0.9; dither technique for setting weighting factor F is used, and values are randomly selected for each generation from the interval $[0.5, 1.0]$; the method for creating trial candidates is $DE/best/1/bin$ (the notation and meaning of all the parameters are explained by Storn and Price [136]).

Weighting Parameter α Used for Calibration

As explained in Section 3.3.1, it is beneficial to use weights $\mathbf{w}^{\mathbf{c}_i}$ of the observed vehicles during the calibration process. These weights are meant to suppress the influence of vehicles whose landmarks were detected inaccurately. Figure 3.9 shows the result of an experiment designed to look for the proper parameter α used for the calibration (3.16). Different parameters α were tested, and for each of them, the plot shows the distribution of the errors (3.4) across all the scenes shown in Figures 3.7 and 3.8. The blue boxplot shows the median value (black central line) and quartiles; the red dotted line in each box shows the average error across all the scenes. Hollow circles show major outliers – scenes that notably failed.

As can be seen from eq. (3.16), parameter α emphasizes vehicles with a smaller re-projection error (3.15) and suppresses vehicles with a higher error. It appears that small values of α lead to instability caused by using the majority of vehicles, which can also contain very noisy

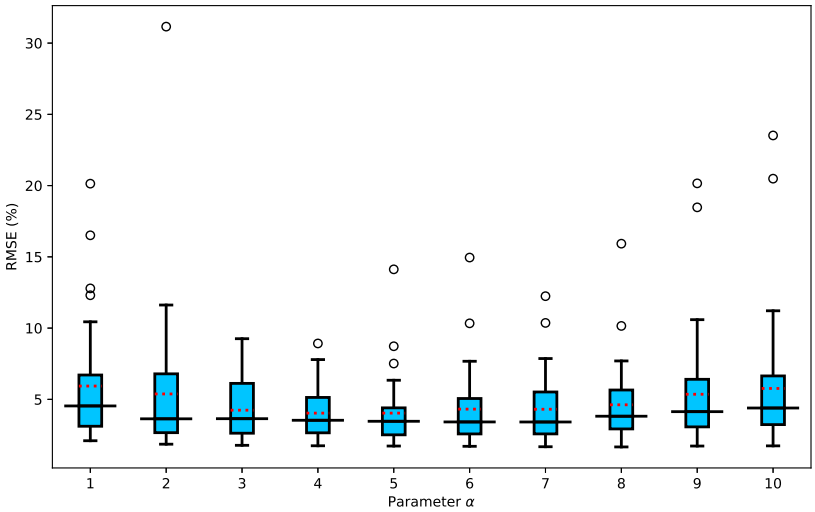

FIGURE 3.9: Calibration error with different values of parameter α used for calibration.

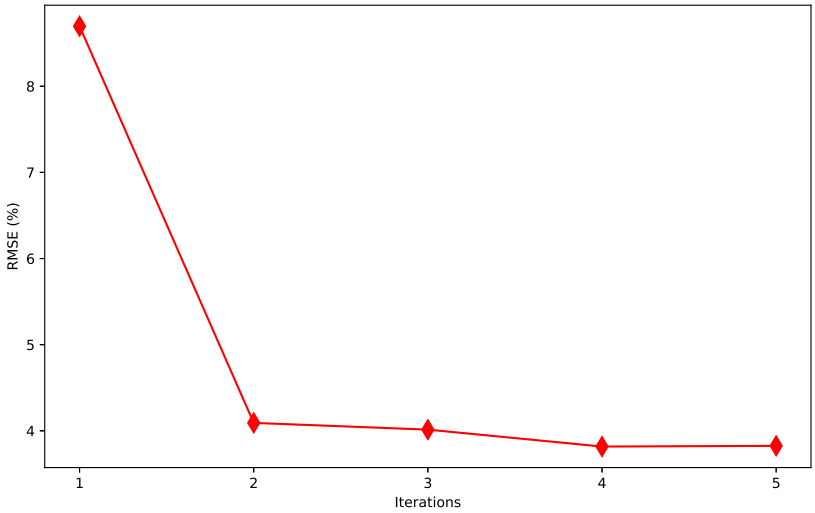

FIGURE 3.10: Error with different count of calibration iterations.

landmarks detections, but on the other hand, large α tends to use very few vehicles with the smallest re-projection error and not exploiting extra information from the other vehicles detected. Therefore in the experiments reported here, $\alpha = 4$ was used. It should be mentioned that the average error across all the scenes is smaller than *AutoCalib* result 6.56 % in all possible tested values of parameter α.

Number of Calibration Iterations

At the end of Section 3.3.1, we described that the whole calibration process is twofold. In the first step, the calibration process is computed with weights $\mathbf{w}^{\mathbf{c}_i}$ set to the value 1.0 and in the second step the weights are computed with more precise focal length value obtained in the first step. We also made an experiment if more iterations with re-computation of the weights with a new value of the focal length are beneficial.

Figure 3.10 shows the result of the experiment; it is apparent that one iteration of calibration with weights set to the value 1.0 produces a much higher error than the usage of multiple

iterations. It seems that a higher number of iterations is not so beneficial, where the largest error reduction is between one and two iterations (about 53 % error reduction); thus, two iterations are used within our experiments. The high reduction of error between one and two iterations also shows the advantage of the weights used, as in the case of one iteration, weights are neglected (set to 1.0).

Study of Calibration Parameters

Till now, we considered an almost perfect camera with the principal point in the middle of the image plane and with no distortion present. However, in real scenarios, cameras can suffer from distortion, and thus we experimented how the proposed method can handle this situation. We extended the described method so that it also estimates the principal point and the distortion parameters. In this setting, intrinsic matrix $\mathbf{K}$ also contains the principal point, and before the computation of 3D distances (3.13), localized 2D landmarks are undistorted by the distortion parameters. Since the experiment should only prove the ability to extend the proposed method by other calibration parameters, only two radial distortion parameters k_1, k_2 are used (different models of camera distortion work with different numbers of parameters). The first two distortion parameters are the most influencing ones [140], and thus these are crucial, and we are not interested in the others. The computation of undistorted points with some distortion parameters can be used as follows:

$$\begin{aligned} x_u &= x_d(1 + k_1 r^2 + k_2 r^4), \\ y_u &= y_d(1 + k_1 r^2 + k_2 r^4), \end{aligned} \tag{3.18}$$

where $r^2 = x_u^2 + y_u^2$, (x_u, y_u) is undistorted point, and (x_d, y_d) is distorted point. These equations can be approximated by an iterative algorithm, and further details can be found in [141, 140].

We tested two settings — in the first one, the precision of calibration was tested on the unchanged original dataset only with an extension of the estimated calibration parameters. The second experiment should test how the method can deal with distortion, and thus localized 2D landmarks in the image plane were distorted by randomly set distortion parameters k_1 and k_2 (in our experiment, random values with uniform distribution from ranges $(-1.0; 1.0)$ and $(0.0; 3.0)$ were used for k_1 and k_2 respectively). As can be seen in Table 3.3, our method seems to be able to handle both cases; both the extended number of parameters and the case when the input data are really distorted. Although the results for the case of extended parameters estimation are slightly better, the main disadvantage of this approach is computational time — 4 additional parameters (p_x, p_y, k_1, k_2) must be estimated, and therefore the computation takes more than twice as much time.

Calibration parameters	Data	Error
f, r_x, r_y, r_z, t_z	Original	4.03 %
f, r_x, r_y, r_z, t_z	Distorted	5.15 %
$f, r_x, r_y, r_z, t_z, p_x, p_y, k_1, k_2$	Original	4.01 %
$f, r_x, r_y, r_z, t_z, p_x, p_y, k_1, k_2$	Distorted	4.05 %

TABLE 3.3: Error of variant with extended parameters and distorted data.

3.4 PlaneCalib: Automatic Camera Calibration by Multiple Observations of Rigid Objects on Plane

In this section, I will describe another method. This time it is not based on distances of landmarks, but the goal is to locate a common ground plane of all observations, which can serve to locate calibration parameters.

3.4.1 Estimation of Extrinsic Camera Parameters

The process of extrinsic camera parameters estimation works with the known intrinsic matrix $\mathbf{K}$; it is thus considered to be known for the purpose of method description. The process of focal length estimation is described later in Section 3.4.3.

Following common notation from Section 3.2.5 — detected 2D ($\bar{\mathcal{K}}^{\mathbf{c}_i}$) and 3D ($\hat{\mathcal{K}}^{\mathbf{c}_i}$) correspondences for each car $\mathbf{c}_i$ can be used to solve the *PnP* [127] problem — the solution provides extrinsic camera parameters (rotation matrix $\mathbf{R}^{\mathbf{c}_i}$ and translation vector $\mathbf{t}^{\mathbf{c}_i}$ for the transformation between the object coordinate system (OCS) and the camera coordinate system (CCS)).

Original 3D positions $\hat{\mathcal{K}}^{\mathbf{c}_i}$ in OCS can thus be transformed into 3D positions in CCS as:

$$\mathbf{k}_j^{\mathbf{c}_i} = [\mathbf{R}^{\mathbf{c}_i}|\mathbf{t}^{\mathbf{c}_i}] \begin{bmatrix} \hat{\mathbf{k}}_j^{\mathbf{c}_i} \\ 1 \end{bmatrix}. \tag{3.19}$$

An example of object points' positions in 3D CCS (after solving *PnP* and transformation by eq. (3.19) from OCS to CCS) with corresponding points in the 2D frame is depicted in Figure 3.11. For each car $\mathbf{c}_i$, parameters for transformation from OCS to CCS are available, and the corresponding point describing the origin of OCS can be transformed into CCS. From at least three of these origin points, a plane in the CCS can be computed (an example can be seen in Figure 3.12).

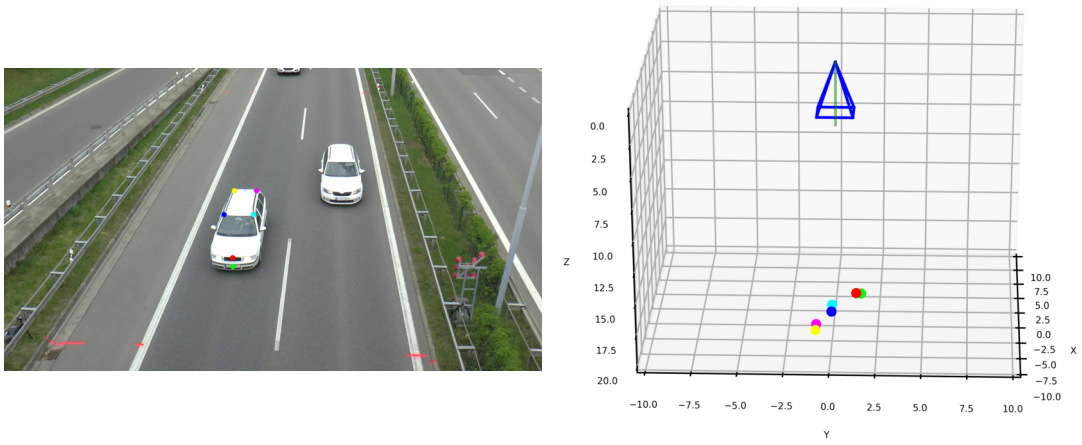

FIGURE 3.11: *left:* Detected 2D landmarks in a single frame. *right:* 3D CCS with corresponding 3D positions of detected landmarks in the 2D frame. The *PnP* solution is computed to obtain parameters $[\mathbf{R}^{\mathbf{c}_i}|\mathbf{t}^{\mathbf{c}_i}]$.

CCS to WCS transformation The extrinsic camera parameters describe the camera position in the WCS and the mutual transformation between WCS and CCS; the world ground plane is

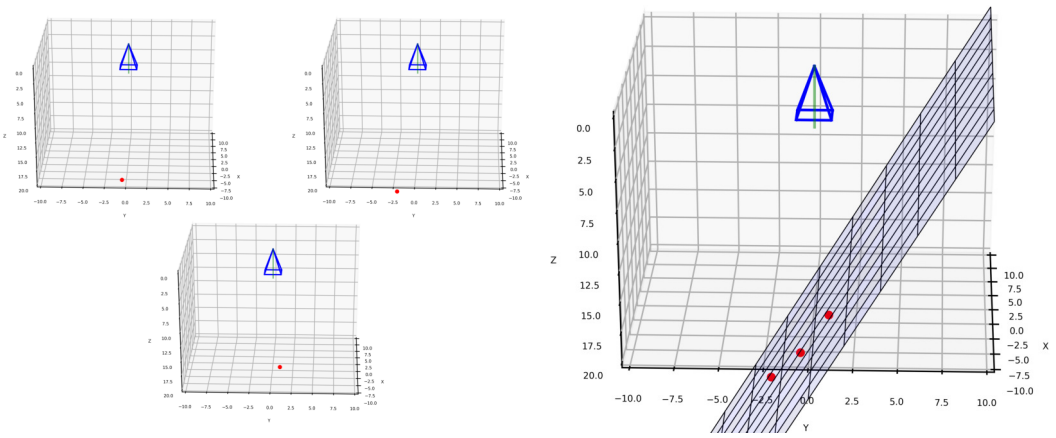

FIGURE 3.12: Computing the plane in the camera coordinate system. *left*: Origins of three different objects transformed into the camera coordinate system. *right*: plane in the camera coordinate system localized by three (minimal case) origins of objects.

the same as the plane in the CCS after applying this transformation — this is the goal of the described method. The equation of the plane is:

$$ax + by + c = z. \tag{3.20}$$

The plane can be recovered in a least squares manner as follows:

$$\begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ & \vdots & \\ x_N & y_N & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_N \end{bmatrix}, \tag{3.21}$$

where x_n, y_n, z_n are 3D coordinates of at least three car's ground points transformed into the camera coordinate system. This equation can be simplified to matrix notation as:

$$A\mathbf{x} = B, \tag{3.22}$$

where $\mathbf{x}$ are the plane parameters $[a, b, c]^\top$. These plane parameters can thus be computed as:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = A^+ B, \tag{3.23}$$

where $A^+ = (A^\top A)^{-1} A^\top$ is Moore-Penrose pseudoinverse of A.

Inference of the plane's normal vector (denoted as $\mathbf{n}$) from parameters $[a, b, c]^\top$ is straightforward (cross-product of two vectors lying on the plane). The rotation matrix $\mathbf{R}$ can be computed from the rotation axis $\mathbf{u}$ and angle θ. The axis is determined by the normal vector $\mathbf{n}$ and camera view vector $\mathbf{c} = [0, 0, 1]^\top$ (marked green in Figures 3.11 and 3.12) as $\mathbf{u} = \mathbf{c} \times \mathbf{n}$. The rotation angle θ is:

$$\theta = \frac{\arccos(\mathbf{c} \cdot \mathbf{n})}{|\mathbf{c}| \cdot |\mathbf{n}|}. \tag{3.24}$$

The final camera rotation matrix $\mathbf{R}$ is then obtained by *Rodrigues* formula:

$$\mathbf{R} = (cos\,\theta)I + (sin\,\theta)[\mathbf{u}]_\times + (1 - cos\,\theta)(\mathbf{u} \otimes \mathbf{u}), \tag{3.25}$$

where $[\mathbf{u}]_\times$ is a cross product matrix (skew-symmetric matrix):

$$[\mathbf{u}]_\times = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix} \tag{3.26}$$

and $\mathbf{u} \otimes \mathbf{u}$ is a tensor product of the vectors:

$$\mathbf{u} \otimes \mathbf{u} = \mathbf{u}\mathbf{u}^\top = \begin{bmatrix} u_x^2 & u_x u_y & u_x u_z \\ u_x u_y & u_y^2 & u_y u_z \\ u_x u_z & u_y u_z & u_z^2 \end{bmatrix}. \tag{3.27}$$

Assuming the world origin in the principal point, the first two elements of the vector $\mathbf{t}$ are set to zero and the Z-coordinate of the camera is derived from equation (3.20) with x and y set to zero (world origin projected into the principal point — no translation within x or y axis). The translation vector $\mathbf{t}$ is then set to $[0, 0, c]^\top$ where c is one of the plane parameters from (3.23). An example of the final calibration plane is shown in Figure 3.13.

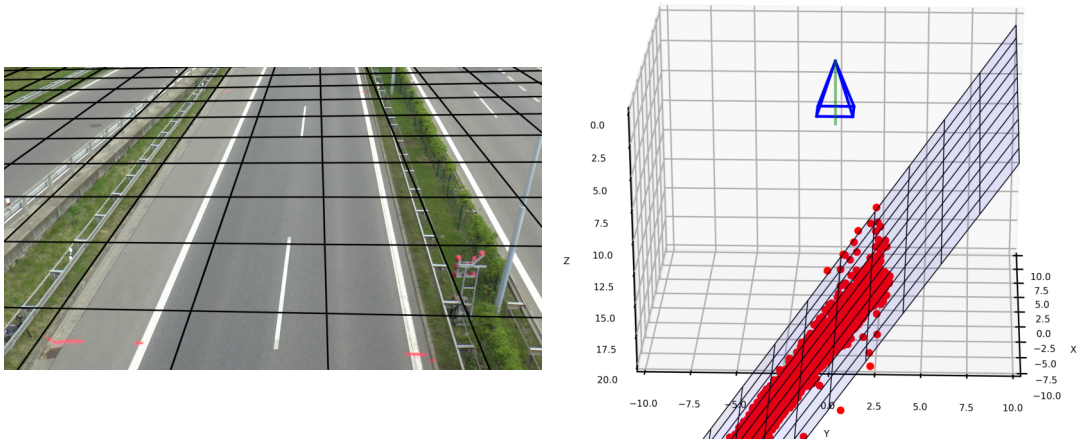

FIGURE 3.13: Final calibration plane with the use of all available cars in $\mathcal{C}$. *left:* 2D frame with the final calibration in WCS. *right:* 3D CCS with potential ground points and fitted plane.

3.4.2 Suppression of Outlier Samples

Similarly to the part describing weights in Section 3.3.1 it is necessary to somehow suppress localization with a high failure rate — there can appear cases when landmarks localization fails significantly in some vehicle instances (occluded ones, weird angles, unusual painting, custom modifications, ...). These outliers bring noise into the plane fitting computation (3.23) and, thereby, the final calibration. For this reason, we propose to compute the re-projection error for each car $\mathbf{c}_i$ and transform it into weight. For each observed car, *PnP* [127] is computed to obtain extrinsic parameters. The 3D points from $\hat{\mathcal{K}}^{\mathbf{c}_i}$ are projected to the image plane by (3.1), yielding again:

$$\tilde{\mathcal{K}}^{\mathbf{c}_i} = \left\{\tilde{\mathbf{k}}_1^{\mathbf{c}_i}, \ldots, \tilde{\mathbf{k}}_K^{\mathbf{c}_i}\right\}. \tag{3.28}$$

The particular vehicle instance $\mathbf{c}_i$ is then assigned its individual normalized re-projection error:

$$\epsilon(\mathbf{c}_i) = \frac{\sum_{j=1}^{K} \left| \tilde{\mathbf{k}}_j^{\mathbf{c}_i}, \bar{\mathbf{k}}_j^{\mathbf{c}_i} \right|}{\sum_{j=1}^{K} \left| \tilde{\mathbf{k}}_j^{\mathbf{c}_i}, \mathbf{K}^{\mathbf{c}_i} \right|}, \tag{3.29}$$

where $\mathbf{K}^{\mathbf{c}_i}$ is the mean of all the points $\bar{\mathbf{k}}^{\mathbf{c}_i}$. This computation is similar to the eq. (3.15).

For each vehicle $\mathbf{c}_i$, its re-projection error $\epsilon(\mathbf{c}_i)$ is thus available, which serves for setting of the *weight*:

$$\mathbf{w}^{\mathbf{c}_i} = \left(\frac{1}{\epsilon(\mathbf{c}_i)} \right). \tag{3.30}$$

Equation (3.23) for fitting a plane to the set of points must be slightly modified for the use of weights. It is convenient to denote the weights as a matrix W with $\mathbf{w}^{\mathbf{c}_i}$ on its diagonal. Then, the plane fitting equation can be modified as:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = (A^\top W A)^{-1} A^\top W B. \tag{3.31}$$

The calibration ground plane is thus computed by the equation (3.31) instead of eq. (3.23), and the rest of the calibration parameters inference is identical to the process described in Section 3.4.1.

3.4.3 Focal Length Estimation

As mentioned earlier, our approach assumes the principal point in the middle of the frame, square pixels, and zero skew. The only unknown intrinsic parameter is thus the *focal length*. Many previous methods, as well as the *AutoCalib* [32] method, assume the focal length to be known — we consider this assumption too limiting as the goal is to calibrate a general camera without any previous knowledge. We thus propose to use focal length estimation.

The process of focal length estimation uses the detected 2D landmarks ($\bar{\mathcal{K}}^{\mathbf{c}_i}$) and precise 3D positions ($\hat{\mathcal{K}}^{\mathbf{c}_i}$) located on the cars' models.

We again define the pairwise distance between points a and b as:

$$\hat{\delta}(\mathbf{c}_i, a, b) = \left| \hat{\mathbf{k}}_a^{\mathbf{c}_i}, \hat{\mathbf{k}}_b^{\mathbf{c}_i} \right|. \tag{3.32}$$

An example of these distances $\hat{\delta}(\mathbf{c}_i, a, b)$ can be seen in Figure 3.6. After using the same process as in eq. (3.9), (3.10), (3.11), (3.12), we get the same distances as in eq. (3.13):

$$\delta(\mathbf{c}_i, a, b, \phi) = \left| \mathbf{k}_a^{\mathbf{c}_i}(\phi), \mathbf{k}_b^{\mathbf{c}_i}(\phi) \right|. \tag{3.33}$$

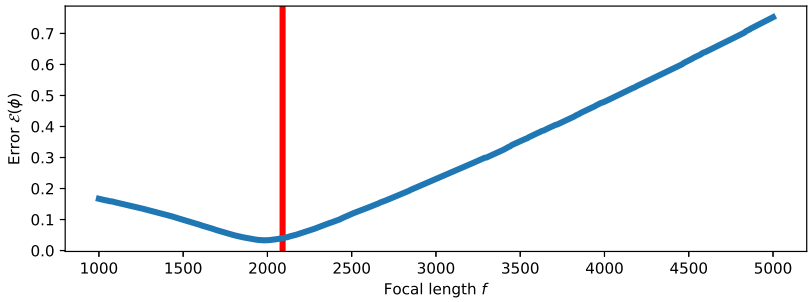

FIGURE 3.14: Error function $\mathcal{E}(\phi)$ on single scene of real world dataset (see Section 3.2). The ground truth focal length value is marked red.

The error of the distance between a pair of visible landmarks given some calibration parameters ϕ can be expressed as follows:

$$\varepsilon(a,b,\phi) = \frac{\delta(\mathbf{c}_i,a,b,\phi) - \hat{\delta}(\mathbf{c}_i,a,b)}{\hat{\delta}(\mathbf{c}_i,a,b)}. \tag{3.34}$$

The total error/cost of all the observations in the scene can be computed as follows:

$$\mathcal{E}(\phi) = \sum_{\mathbf{c}_i \in \mathcal{C}} \sum_{a,b} \frac{\mathbf{w}^{\mathbf{c}_i} \varepsilon(a,b,\phi)}{\mathbf{w}^{\mathbf{c}_i}}, \tag{3.35}$$

where $\mathbf{w}^{\mathbf{c}_i}$ is the weight defined by (3.30). The process of focal length estimation consists of localizing such a focal length f that minimizes this error (3.35). Once a focal length value f is determined, the rest of the calibration parameters ($\mathbf{R}_\phi$ and $\mathbf{t}_\phi$) are determined by the method described in Section 3.4.1.

As can be seen in Figure 3.14, the proposed error function is convex, and a single minimal value can be found. In our experiments, the optimal focal length f is found by *Brent method*.

Summary of Calibration Process

The main part of the whole calibration is the optimization of focal length f. In each step of the focal length optimization, *weights* (as defined by (3.30)) are computed as these can change by the different settings of parameter f. Afterward, extrinsic camera parameters are obtained by the process described in Section 3.4.1. With all known camera parameters ϕ, the error function (3.35) can be evaluated. Optimization reaches the best possible focal length value f together with the final extrinsic camera calibration parameters.

3.4.4 Results

As mentioned in Section 3.2.4 RMSE is used to evaluate the method. The method was compared to *OptInOpt* method (Section 3.5) because it was released before this method and it was possible to make the comparison publicly available. This comparison can be seen in Figure 3.15. The mean RMSE across all scenes is 3.65 % for the *PlaneCalib* method and 3.01 % for the *OptInOpt* method (median values 3.23 % and 2.43 % for *PlaneCalib* and *OptInOpt* respectively). Several examples of calibration on the *BrnoCompSpeed* dataset can be seen in Figure 3.16.

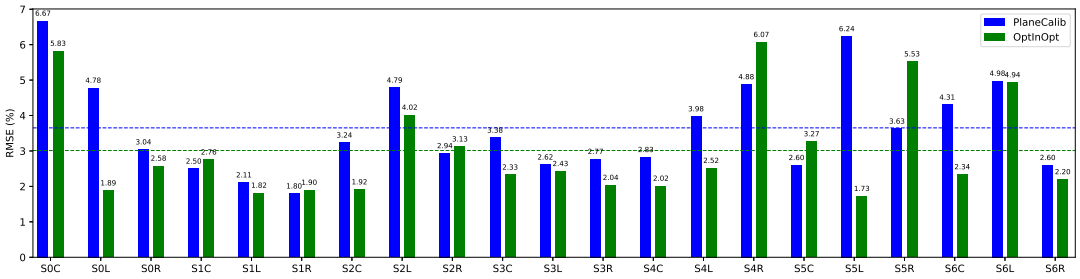

FIGURE 3.15: Comparison of accuracy for the proposed *PlaneCalib* method and the *OptInOpt* method. Average results are as follows: *PlaneCalib* 3.65 %, *OptInOpt* 3.01 %. These results are measured when calibrating from all the input samples; further, a mechanism for the selection of a more suitable subset is discussed.

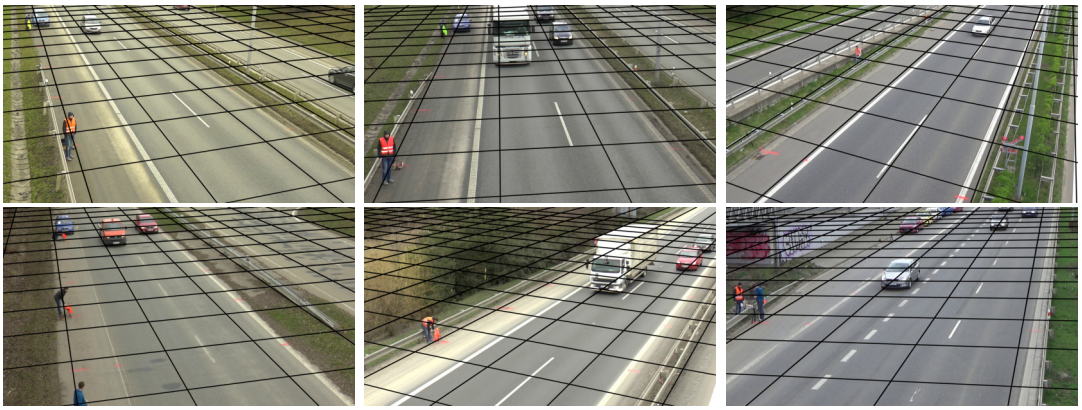

FIGURE 3.16: Sample images from *BrnoCompSpeed* dataset [135] together with resulting calibration.

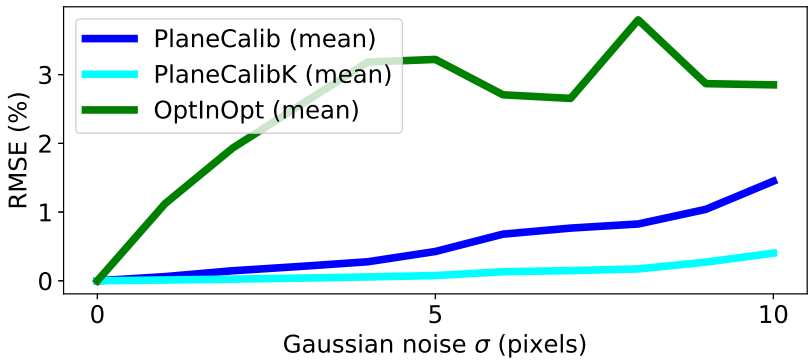

FIGURE 3.17: Comparison of the calibration error by the proposed *PlaneCalib* method and the *OptInOpt* method on synthetic dataset w.r.t. varying noise levels in landmarks' 2D positions. Random Gaussian shift is added to all localized landmarks — noise in landmarks localization.

Noise in Landmarks Detection The influence of the inaccuracies in the detection of the landmarks was tested on the synthetic dataset (Section 3.2.3). Random values from the *Gaussian distribution* were added to all detected 2D landmarks, which simulates the error in the localization of 2D landmarks. The results of calibration accuracy w.r.t. varying noise levels can be seen in Figure 3.17. Similarly to the evaluation on the real dataset, RMSE error (3.4) of the known distances in the scene was computed.

The *PlaneCalib* method uses focal length estimation (Section 3.4.3), *PlaneCalibK* counts with a known intrinsic matrix $\mathbf{K}$. The *PlaneCalib* method seems to be more resistant to noise in the landmarks' positions than the *OptInOpt* method. The results presented are the mean values of 100 trials carried out using random scenes from the synthetic dataset.

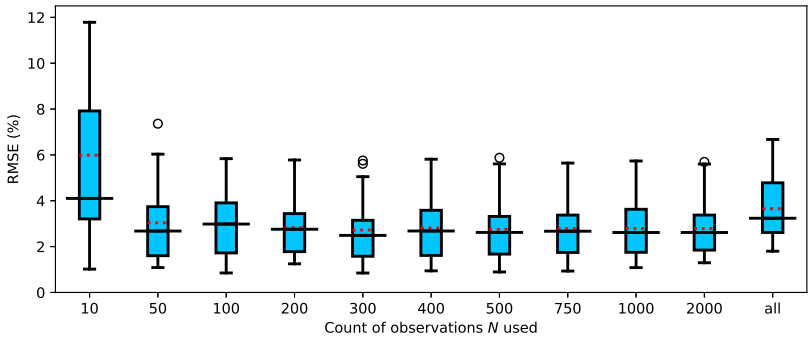

FIGURE 3.18: Calibration error on all scenes from *BrnoCompSpeed* dataset with a varying count of observations used.

Computation Speed-up and Improvement To speed up and improve the calibration process, an approach that does not use all the observations was considered. As was mentioned in Section 3.4.3, in each optimization step, the weights of individual observations are computed. To make the calibration process faster, after each re-computation of the weights, only N best observations (those with the highest weights) are used for further computation. The influence of value N selection can be seen in Figure 3.18. In cases with high N, some noisy observations can be present during calibration and these can affect the result. On the other hand, cases with low N can become problematic due to insufficient observation coverage of the ground plane. During our experiments, all observations in the scene were used for a fair comparison with *OptInOpt*. But with the proper selection of value N, the resulting accuracy of 2.72 % error can be reached, which outperforms the *OptInOpt* method.

3.5 OptInOpt: Dual Optimization for Automatic Camera Calibration by Multi-Target Observations

Another proposed method is again somehow different from others. In this case, the locations of single vehicles are optimized due to some calibration parameters, which are also optimized. The proposed method is based on dual optimization — in each step of camera parameters optimization are optimized positions of all single detected objects. These objects must fit the position of localized landmarks as precisely as possible.

3.5.1 Optimization of Object Position

Following previous notation (Section 3.2.5) each detected vehicle $\mathbf{c}_i$ is equipped with 3D positions of detected landmarks $\hat{\mathcal{K}}^{\mathbf{c}_i}$. However, these 3D positions are in the object coordinate system and thus, no relations between different observations are not available by these positions. For this reason, the positions of single observed vehicles must be optimized. Let us assume some calibration parameters $\mathbf{K}_\phi$, $\mathbf{R}_\phi$ and $\mathbf{t}_\phi$, whose estimation is described later in Section 3.5.3. The position of an object in the world space is described by some transformation. In our case, it is assumed that all objects lie within the common ground plane, and thus their

position in the space is described only by rotation around the Z-axis and translation along X- and Y-axis (potentially more parameters can be used leading to an extension — relaxing the constraint of objects' common plane).

An object's position can be transformed by a transformation matrix:

$$\mathbf{M}(\omega_\phi) = \begin{bmatrix} cos(r_z) & -sin(r_z) & 0 & t_x \\ sin(r_z) & cos(r_z) & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.36}$$

to the world coordinate system by transformation parameters ω_ϕ, consisting of Z-axis rotation r_z, X-axis translation t_x and Y-axis translation t_y as:

$$\hat{\mathbf{k}}_j^{\mathbf{c}_i}(\omega_\phi) = \mathbf{M}(\omega_\phi) \begin{bmatrix} \hat{\mathbf{k}}_j^{\mathbf{c}_i} \\ 1 \end{bmatrix}. \tag{3.37}$$

3D points $\hat{\mathcal{K}}^{\mathbf{c}_i}(\omega_\phi)$ transformed by some transformation parameters ω_ϕ can be projected into 2D image plane by using eq. (3.1) as $\tilde{\mathcal{K}}^{\mathbf{c}_i}(\omega_\phi)$. The resulting position of a single object can thus be found by optimizing parameters ω_ϕ to minimize the re-projection error $\epsilon(\bar{\mathcal{K}}^{\mathbf{c}_i}, \tilde{\mathcal{K}}^{\mathbf{c}_i}(\omega_\phi))$ defined by (3.38). Our experiments show that this minimization problem is non-convex, and thus a global optimization method must be used. In all our experiments, *differential evolution* [136] algorithm was used due to its computational speed and stability (with suitably selected bounds for searching for the solution). An example of optimized vehicle positions can be seen in Figure 3.19.

FIGURE 3.19: *left:* Projected 3D landmarks' positions $\hat{\mathcal{K}}^{\mathbf{c}_i}$ to the image plane (green points) and localized landmarks positions $\bar{\mathcal{K}}^{\mathbf{c}_i}$ (red points). *right:* Projected 3D landmarks' positions $\hat{\mathcal{K}}^{\mathbf{c}_i}(\omega_\phi)$ (green points) after optimization of transformation parameters ω_ϕ and localized landmarks positions $\bar{\mathcal{K}}^{\mathbf{c}_i}$ (red points).

3.5.2 Estimation of Objects Weight

Similarly to processes described in Sections 3.3.1 and 3.4.2 in this case are again estimated weights of single detected vehicles. For each vehicle $\mathbf{c}_i$, calibration parameters by *PnP* [127] can be computed as 2D correspondences $\bar{\mathcal{K}}^{\mathbf{c}_i}$ and 3D correspondences $\hat{\mathcal{K}}^{\mathbf{c}_i}$ are known. With known calibration parameters, projected points $\tilde{\mathcal{K}}^{\mathbf{c}_i}$ can be computed for each $\hat{\mathcal{K}}^{\mathbf{c}_i}$ by eq. (3.1) as it serves for projection to the image plane. With known projected points $\tilde{\mathcal{K}}^{\mathbf{c}_i}$ and localized

landmarks $\bar{\mathcal{K}}^{\mathbf{c}_i}$, the re-projection error $\epsilon(\mathbf{c}_i)$ can be computed as:

$$\epsilon(\bar{\mathcal{K}}^{\mathbf{c}_i}, \tilde{\mathcal{K}}^{\mathbf{c}_i}) = \sum_j \left\| \bar{\mathbf{k}}_j^{\mathbf{c}_i}, \tilde{\mathbf{k}}_j^{\mathbf{c}_i} \right\|^2. \tag{3.38}$$

Since the *PnP* needs the intrinsic matrix $\mathbf{K}$, which is assumed to be unknown, re-projection error $\epsilon(\bar{\mathcal{K}}^{\mathbf{c}_i}, \tilde{\mathcal{K}}^{\mathbf{c}_i})$ is computed repeatedly F times with randomly set focal length f (from a reasonable interval and with uniform distribution) and principal point in the middle of the image plane. By this approach, F possible re-projection errors are available for each vehicle $\mathbf{c}_i$ and their mean $\overline{\epsilon(\bar{\mathcal{K}}^{\mathbf{c}_i}, \tilde{\mathcal{K}}^{\mathbf{c}_i})}$ is used further. For each vehicle $\mathbf{c}_i$, the weight is thus computed as the reciprocal value of the re-projection error:

$$w(\mathbf{c}_i) = \frac{1}{\overline{\epsilon(\bar{\mathcal{K}}^{\mathbf{c}_i}, \tilde{\mathcal{K}}^{\mathbf{c}_i})}}. \tag{3.39}$$

3.5.3 Optimization of Calibration Parameters

The goal of camera calibration is to find such parameters that best describe the camera setting and position in the world. Camera calibration parameters will be denoted as ϕ and they consist of camera intrinsic matrix $\mathbf{K}_\phi$, camera rotation matrix $\mathbf{R}_\phi$, and camera translation vector $\mathbf{t}_\phi$. For each possible setting of parameters ϕ, world positions of the objects $\hat{\mathcal{K}}^{\mathbf{c}_i}(\omega_\phi)$ are available, as was described in previous Section 3.5.1. For each car $\mathbf{c}_i$, the re-projection error $\epsilon(\bar{\mathcal{K}}^{\mathbf{c}_i}, \hat{\mathcal{K}}^{\mathbf{c}_i}(\omega_\phi))$ can also be computed.

Similarly to *PnP* algorithms which determine the calibration parameters for one object based on 2D-3D correspondences, our solution is based on the minimization of the re-projection error. The objective function which is being minimized is as follows:

$$\varepsilon(\phi) = \frac{\sum_{\mathbf{c}_i \in \mathcal{C}} \epsilon\left(\bar{\mathcal{K}}^{\mathbf{c}_i}, \hat{\mathcal{K}}^{\mathbf{c}_i}(\omega_\phi)\right) w(\mathbf{c}_i)}{\sum_{\mathbf{c}_i \in \mathcal{C}} w(\mathbf{c}_i)}, \tag{3.40}$$

which is a weighted arithmetic mean of re-projection errors of all detected vehicles $\mathcal{C}$, with weights computed as described in Section 3.5.2. The objective function is again non-convex and *differential evolution* algorithm is used for its solving. In each iteration of the optimization, some calibration parameters ϕ are set, and these parameters are used for the optimization of objects' positions in the world coordinate system $\hat{\mathcal{K}}^{\mathbf{c}_i}(\omega_\phi)$. Optimization iterations of parameters ϕ tend to minimize the proposed objective function (3.40). Potentially any global optimization algorithm can be used instead, but *differential evolution* seems to be robust and provides good results.

3.5.4 Results

Because the proposed method was published before *PlaneCalib* method and also before *BrnoCarPark* dataset, it is compared only with *AutoCalib* on *BrnoCompSpeed* dataset and on synthetic dataset 3.2.3.

In the case of optimization of objects' positions, the population size and iteration values are set to 25 and 200, respectively. For the optimization of calibration parameters, the population

size and iteration values are set to 20 and 500, respectively. The value for objects' weights computation F, as described in Section 3.5.2, is set to the value of 10.

Comparison of *OptInOpt* and *AutoCalib* methods is shown in Figure 3.20. The mean RMSE (3.4) across all scenes and all observations is 2.85 % for *OptInOpt* and 6.88 % for *AutoCalib* (median values 2.50 % and 5.49 % for *OptInOpt* and *AutoCalib*, respectively). It can also be seen that *OptInOpt* seems to be more stable without extreme outlier values, which appear in the case of *AutoCalib*; moreover, it achieves better results without the knowledge of the focal length.

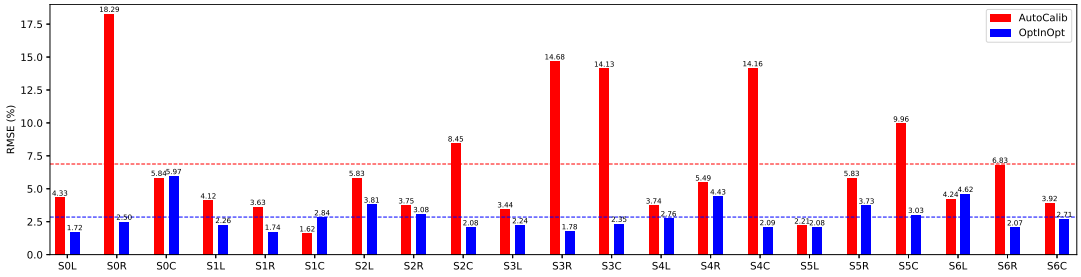

FIGURE 3.20: Comparison of the calibration accuracy for the proposed *OptInOpt* method and *AutoCalib* method. Average results are as follows: *OptInOpt* 2.85 %, *AutoCalib* 6.88 %.

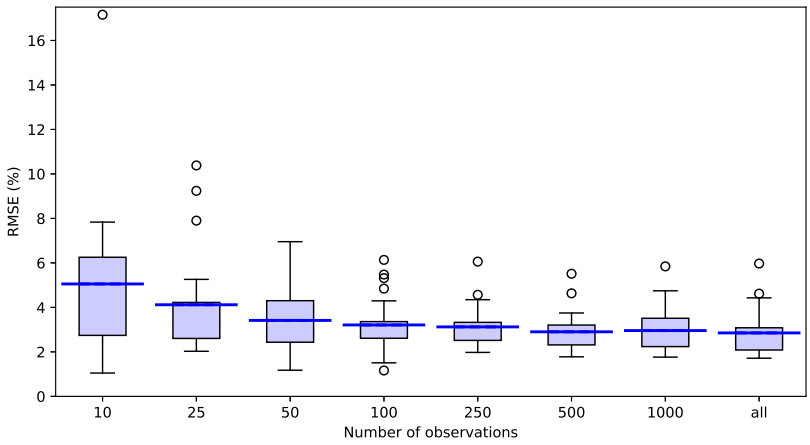

FIGURE 3.21: Proposed algorithm accuracy w.r.t. different count of observations used. Results for all scenes from *BrnoCompSpeed* dataset (mean values denoted by blue line).

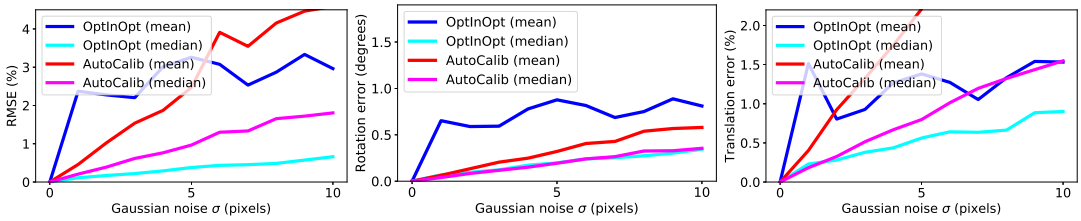

FIGURE 3.22: Calibration accuracy by the proposed method *OptInOpt* and by *AutoCalib* on synthetic dataset w.r.t. varying noise levels in 2D positions of landmarks. Random Gaussian shift with standard deviation σ is added to all 2D landmarks' positions. *left*: Calibration error, *center*: rotation error, *right*: translation error.

The relation of calibration accuracy w.r.t. the count of observations used was also evaluated. The results can be seen in Figure 3.21. In this experiment, N observations with the highest weight $w(\mathbf{c}_i)$ were used for each scene from the *BrnoCompSpeed* dataset. It is apparent that the increasing number of observations leads to better calibration accuracy — option *all* contains all observations for each scene.

Other experiments were evaluated on the synthetic dataset (Section 3.2.3). The noise influence on the landmarks' localization was experimented with. The results of these experiments w.r.t. varying noise levels can be seen in Figure 3.22. A random shift from Gaussian distribution with varying standard deviation σ was added to 2D positions of landmarks — RMSE, rotation, and translation errors are visualized. In all cases, *OptInOpt* seems to be at least comparable or outperforms *AutoCalib*, moreover without the knowledge of focal length. The presented values are the results of 500 trials over random scenes from the synthetic dataset.

3.6 Evaluation and Combinations of Proposed Methods

As the methods were published gradually it was not possible to make comparisons and evaluations between all the methods. Every time was made a comparison with a suitable method and available datasets; thus in previous Sections 3.3, 3.4, and 3.5 evaluations were often limited by the time when the relevant paper was written — used results are from papers as the text of these sections is based on the papers.

After publishing all the proposed methods it is now possible to evaluate all methods together and on both relevant datasets — *BrnoCarPark* 3.2.2 as well as *BrnoCompSpeed* 3.2.1. For comparison also *AutoCalib* method was evaluated on both datasets and the result can be seen in Figure 3.23. The result of *LandmarksCalib*, *PlaneCalib*, and *OptInOpt* methods can be seen in Figures 3.24, 3.25, and 3.26 respectively. It is necessary to emphasize that as all methods somehow rely on optimization each run can result in slightly different results and thus results can differ a bit compared to the values presented in papers.

From the graph is apparent that the *BrnoCarPark* (right side of graphs) is more difficult for all the methods. There appear some extreme "failure cases" in methods *AutoCalib* and *PlaneCalib*; also some not extreme "failure cases" appear in *OptInOpt* method. The *LandmarksCalib* seems to be the most stable and reaches good results on all the scenes. The exact values are available in Table 3.4.

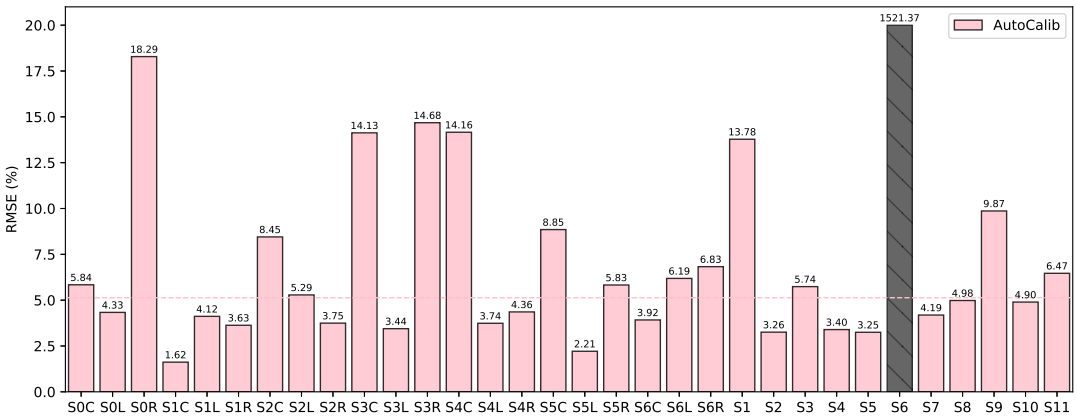

FIGURE 3.23: *AutoCalib* method results

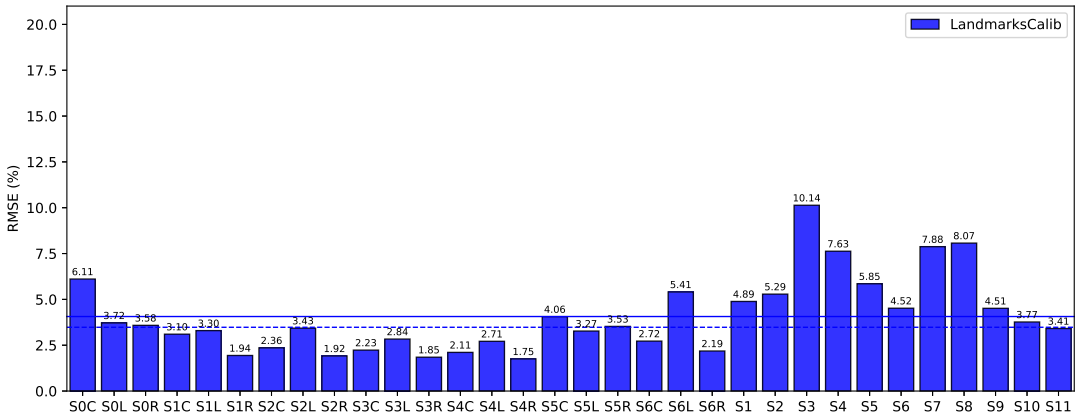

FIGURE 3.24: *LandmarksCalib* method results

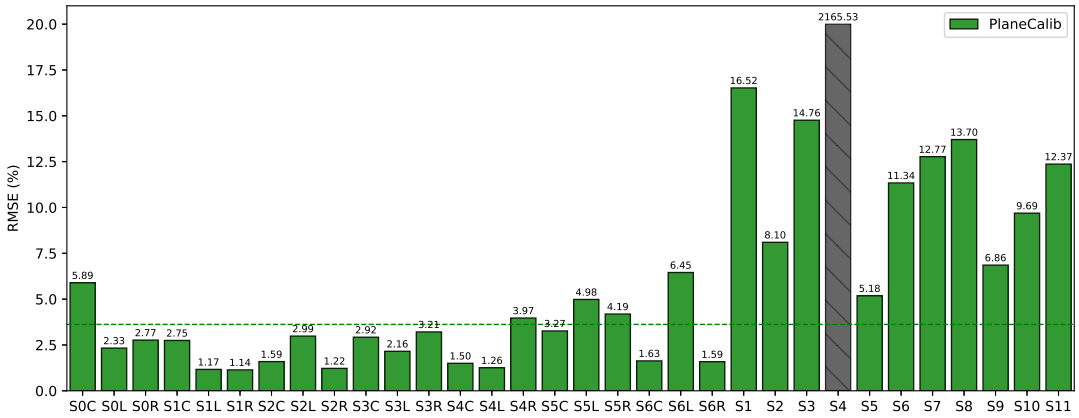

FIGURE 3.25: *PlaneCalib* method results

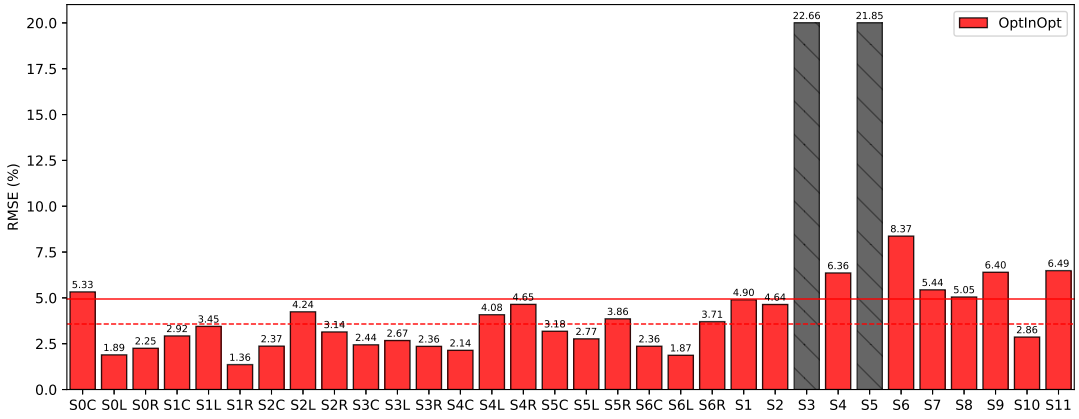

FIGURE 3.26: *OptInOpt* method results

Combination of Methods

For the next experiment, I tried to combine the methods. The idea was that, for example, *PlaneCalib* can make some coarse assumptions of calibration parameters and methods which use optimization (*LandmarksCalib*, *OptInOpt*) can then make optimization in limited parameter space. The last part of combinations is thus every time method using optimization; optimization then proceeds in smaller bounds of parameters compared to the method as is.

Method	Mean	Median
AutoCalib	53.9	5.13
LandmarksCalib	4.06	3.47
PlaneCalib	72.99	3.62
OptInOpt	4.93	3.57
Combination 1	4.62	3.85
Combination 2	5.46	4.53
Combination 3	17.67	10.33
Combination 4	34.20	26.97
Combination 5	6.92	4.42
Combination 6	7.91	5.70

TABLE 3.4: Evaluation of all methods (RMSE)

Possible combinations are as follows:

- **Combination 1**: *PlaneCalib* → *LandmarksCalib*
- **Combination 2**: *PlaneCalib* → *OptInOpt*
- **Combination 3**: *LandmarksCalib* → *OptInOpt*
- **Combination 4**: *OptInOt* → *LandmarksCalib*
- **Combination 5**: *PlaneCalib* → *LandmarksCalib* → *OptInOpt*
- **Combination 6**: *PlaneCalib* → *OptInOpt* → *LandmarksCalib*

The graphs of combinations' evaluation can be see in Figures 3.27 (Combination 1), 3.28 (Combination 2), 3.29 (Combination 3), 3.30 (Combination 4), 3.31 (Combination 5), and 3.32 (Combination 6). The exact values are available in Table 3.4. From the result is apparent that neither combination brings any improvement in reached values. Another finding is a failure case when two "optimization" methods are combined together. It is probably caused by lower bounds for the localization of parameters in a second step (method). On the other hand, neither combination failed significantly when *PlaneCalib* is the first in combination.

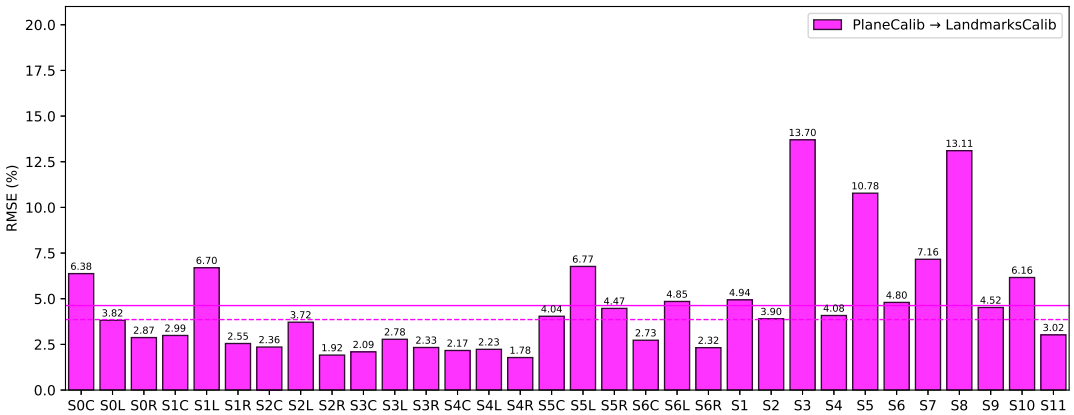

FIGURE 3.27: **Combination 1**: *PlaneCalib* → *LandmarksCalib*

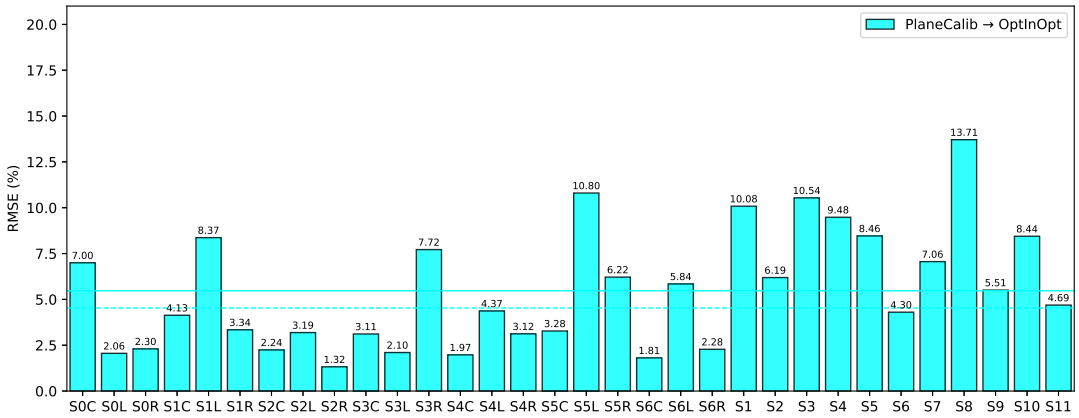

FIGURE 3.28: **Combination 2**: *PlaneCalib → OptInOpt*

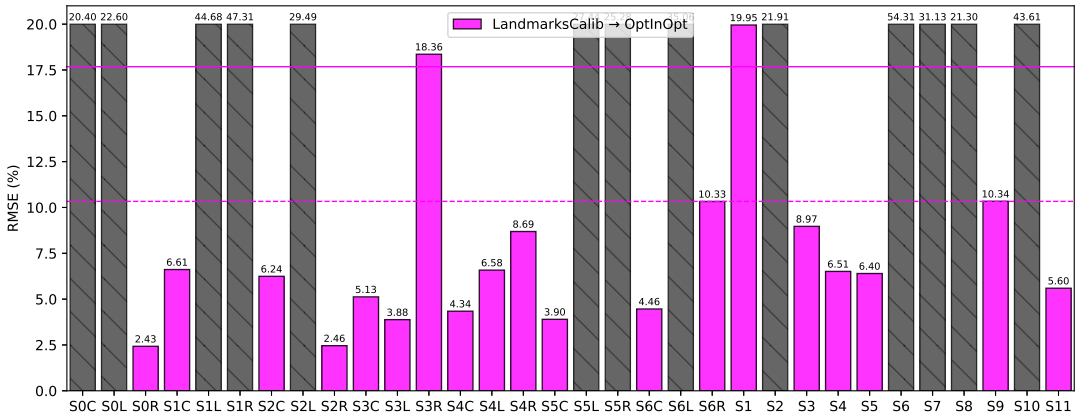

FIGURE 3.29: **Combination 3**: *LandmarksCalib → OptInOpt*

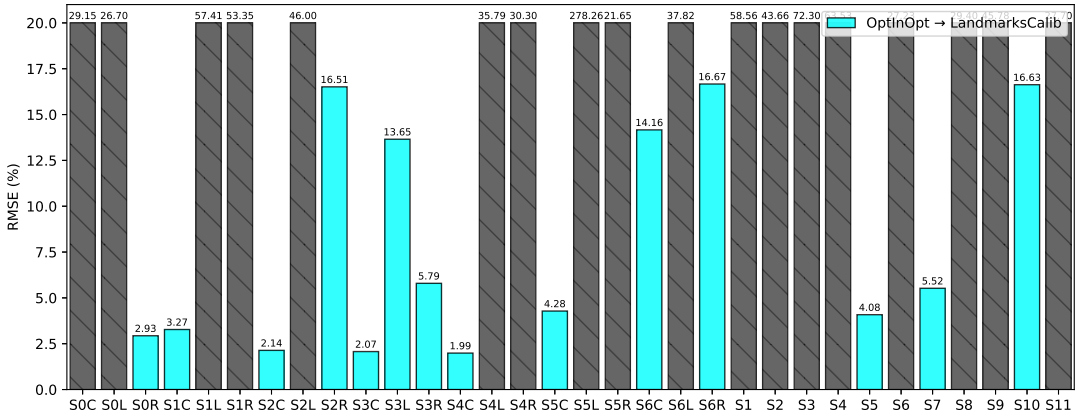

FIGURE 3.30: **Combination 4**: *OptInOt → LandmarksCalib*

Computation Time

Another experiment with computation time was done. I was interested in how the count of observations in the scene affects the computation time. In graphs 3.33 (*LandmarksCalib*), 3.34 (*PlaneCalib*), and 3.35 (*OptInOpt*) are depicted times of computation for each method. As expected, *OptInOpt* method is extremely slow, compared to other methods, due to its dual optimization. Another expectation was the "more linear" process of *PlaneCalib* method because of less usage of optimization methods. On the other hand, *LandmarksCalib* and

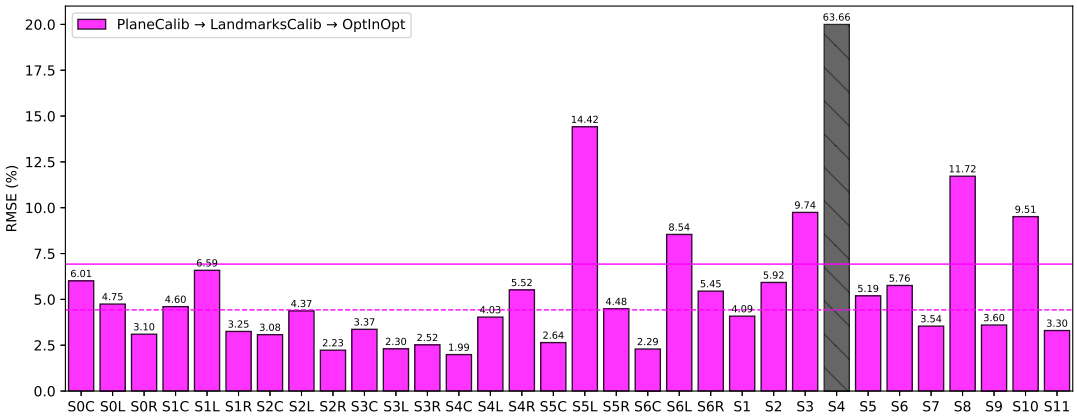

FIGURE 3.31: **Combination 5**: *PlaneCalib → LandmarksCalib → OptInOpt*

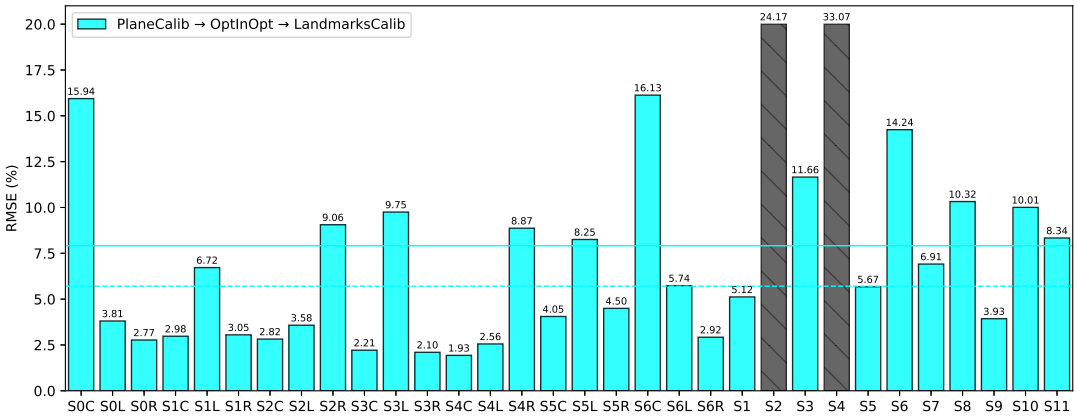

FIGURE 3.32: **Combination 6**: *PlaneCalib → OptInOpt → LandmarksCalib*

OptInOpt methods use optimization and can finish computation much faster or slower — can get stuck in a local minimum.

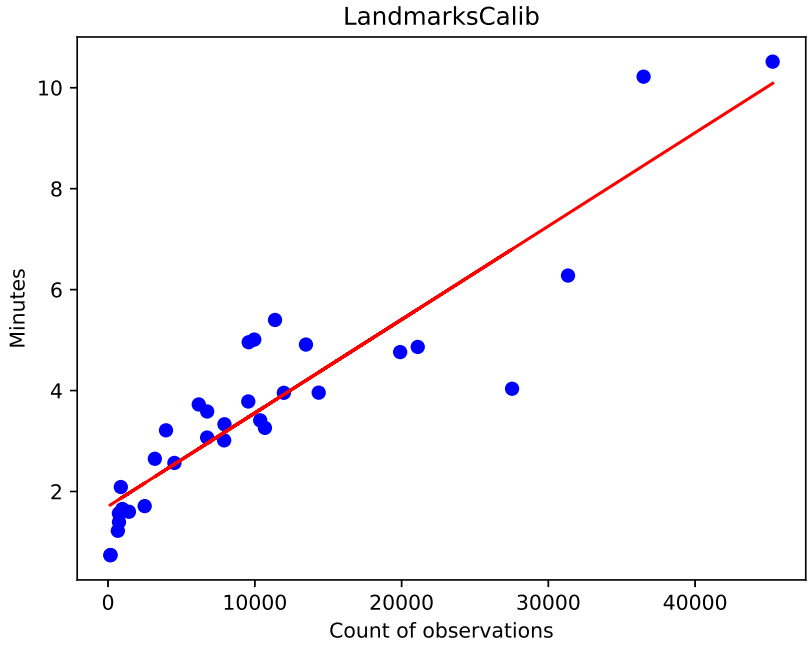

FIGURE 3.33: Computation time of *LandmarksCalib* method

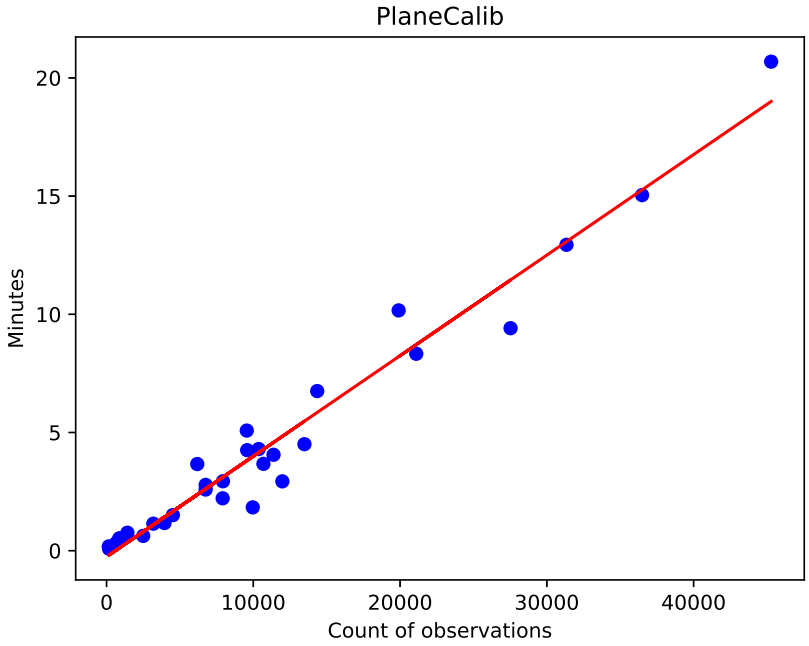

FIGURE 3.34: Computation time of *PlaneCalib* method

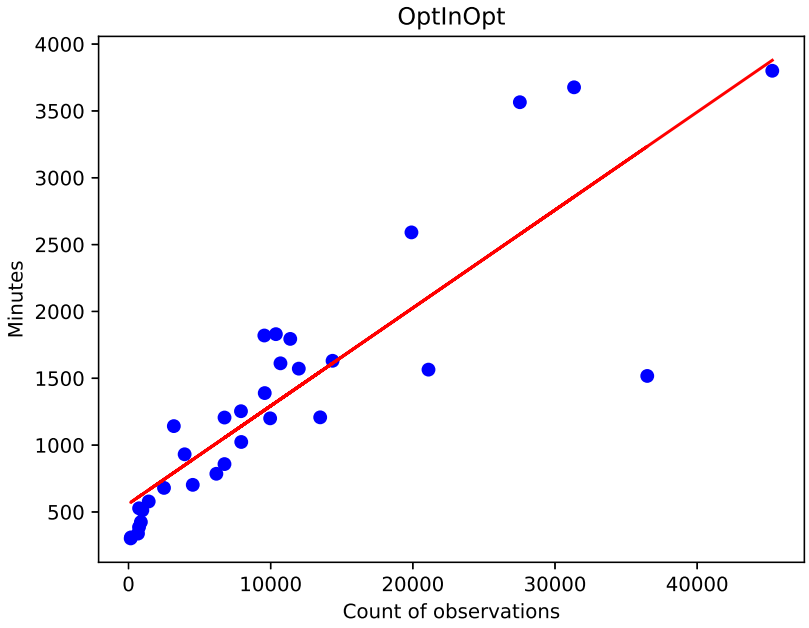

FIGURE 3.35: Computation time of *OptInOpt* method

Chapter 4

Participation in Challenges

In this section, I want to mention some information about our participation in challenges over the years. As I mentioned earlier, I turned the aim of my work more to ITS as my colleagues participated in *AI City Challenge* in 2018 [142], and it seemed interesting to me. So I added to my colleagues in 2019 [7] in participation to Tracks *City-Scale Multi-Camera Vehicle Tracking* and *City-Scale Multi-Camera Vehicle Re-Identification*. The goal is to track and recognize single vehicles traveling through a city. Cameras' views can be overlapping but also can be a few kilometers far from each other. Once, we also participated in *Multi-Class Product Counting & Recognition for Automated Retail Checkout*, where the goal is to recognize which products are placed under the camera view. It is a step to entirely self-service checkout — there are problems with overlapping single products so as overlapping by human hands.

4.1 AI City Challenge — Vehicle Re-Identification

Here is a copy of our paper [7] as a submission to *AI City Challenge*. I decided to place this paper here as I consider this paper more interesting than the second one [8]. In this paper, I made a big portion of the work and proposed some important parts of new ideas. We try some new approaches with positional matching, which we still have in mind, and potentially we will sometimes evolve this approach.

4.1.1 Abstract

In our submission to the NVIDIA AI City Challenge, we address vehicle re-identification and vehicle multi-camera tracking. Our approach to vehicle re-identification is based on the extraction of visual features and aggregation of these features in the temporal domain to obtain a single feature descriptor for the whole observed track. For multi-camera tracking, we proposed a method for matching vehicles by the position of trajectory points in real-world space (linear coordinate system). Furthermore, we use CNN for the vehicle re-identification task to filter out false matches generated by the proposed **positional matching** method for better results.

4.1.2 Introduction

In this submission, we address the tasks of vehicle multi-camera tracking and re-identification of the NVIDIA AI City Challenge 2019 (*i.e. Track1* and *Track2*).

Our approach to visual vehicle re-identification is based on the extraction of feature vectors using a convolutional neural network and aggregation of extracted feature vectors from observed vehicles in the temporal domain. We use standard CNNs [74, 143, 75] trained for the identification task and we employ an LFTD network [5] for feature aggregation.

For the multi-camera tracking part, we propose a method for matching points from vehicle trajectories in real-world linear coordinate system space. This approach is based on a projection of 2D image points into the real-world linear space [144] and the matching of vehicles in this linear space with respect to time and space constraints. Furthermore, this approach can be also combined with the extraction of feature vectors for all observed and pre-matched tracks.

To put our approach in a larger context, we include a brief overview of the state of the art in vehicle re-identification. After that, we describe the used methods for both vehicle re-id and multi-camera tracking in detail.

Vehicle Re-Identification

Formerly, the methods for vehicle re-identification were based on automatic license plate recognition [145, 146, 147], using hand-crafted visual features (PCA-SIFT, HOG descriptors, color histograms, *etc.*) extracted from vehicle images [148, 149, 150] or just information about the date, time, color, speed and vehicles' dimensions [149].

Recently, *deep* features learned by CNNs [151, 152, 153, 154, 155] are being used for this task. Liu *et al.* [156] combine the hand-crafted and deep features. Improvements were also made by exploiting *spatio-temporal* [156, 153] or *visual-spatio-temporal* [152] properties. Some of them benefit from Siamese CNNs for license plate verification [156] or vehicle image similarities [152]. Moreover, introduction of triplet loss [155, 157] or Coupled Cluster Loss (CCL) [151] led to accuracy improvements and faster convergence. Recently, Yan *et al.* [154] proposed to use Generalized Pairwise Ranking or Multi-Grain based List Ranking for retrieval of similar vehicles, which performs even better than CCL.

A few person re-identification papers also proposed to use the triplet loss [158, 159] or quadruplet loss [160] instead of training the network in a Siamese setting. There were also attempts to learn a metric for the re-identification like KISSME [161], XQDA [162], You *et al.* [53] learn Mahalanobis distance on LBP and HOG3D features, and finally Shi *et al.* [163] learn Mahalanobis distance in an end-to-end manner.

On the other hand, a group of methods exists that propose to use feature pooling (aggregation) in a temporal domain for re-identification tasks. Such pooling is usually used in the context of person re-identification (with the exception of Yang *et al.* [164], who used it for video face recognition). The methods are often trained by using a Siamese network [165, 166, 167, 168, 169, 164] with contrastive loss and optionally the identification loss as well. Similarly, in our paper [5], we propose a method for feature aggregation in the temporal domain of multiple observations of a vehicle in one track.

Vehicle Re-Identification Datasets

Comprehensive datasets of vehicles for fine-grained vehicle recognition are available [170, 171, 135] for more than 5 years now. However, when it comes to vehicle re-identification the available datasets are limited in some ways. Liu *et al.* [156] constructed a rather small VeRi-776 dataset containing 50 000 images of 776 vehicles. Liu *et al.* [151] collected VehicleID dataset containing 26 267 vehicles in 220k images taken from a frontal/rear viewpoint above the road. Recently, Yan *et al.* [154] published two datasets VD1 and VD2 for vehicle re-identification and fine-grained classification with over 220k of vehicles in total, with make, model, and year annotation. However, both datasets are limited to frontal viewpoints only.

Recently, we collected dataset **CarsReId74k** [5], which contains ≈74k of vehicle tracks from various viewpoints with precise ground truth identity acquired from a zoomed-in camera by license plate recognition.

4.1.3 Used Approach

In our submission to the NVIDIA AI City Challenge 2019, we focused on vehicle re-identification (Track 2) and vehicle multi-camera tracking (Track 1). In the following text, we describe our approach to both of these tasks.

FIGURE 4.1: Video screenshots from one recording session with a vehicle with the same identity. Image source: [5].

Training Data for Vehicle Re-Identification

Both tasks contain vehicles observed from various viewpoints. It is necessary to acquire a similar dataset for pre-training of the identification and also re-identification networks. We used our dataset CarsReId74k [5] which contains 17 681 unique vehicles, 73 976 observed tracks, and 292 226 positive pairs. For examples of positive and negative pairs, see Figure 4.2.

The dataset was collected using 8 cameras recording at the same time. Four cameras always observed the same direction of traffic at one location from different viewpoints (left, center, right), and one camera was zoomed in, and it was used for license plate detection and recognition by our recent method [172]. The videos at one location were approximately synchronized and the recognized license plates were assigned to the detected vehicles from other cameras, producing the identities for all the vehicles. See Figure 4.1 for examples of videos from one recording session.

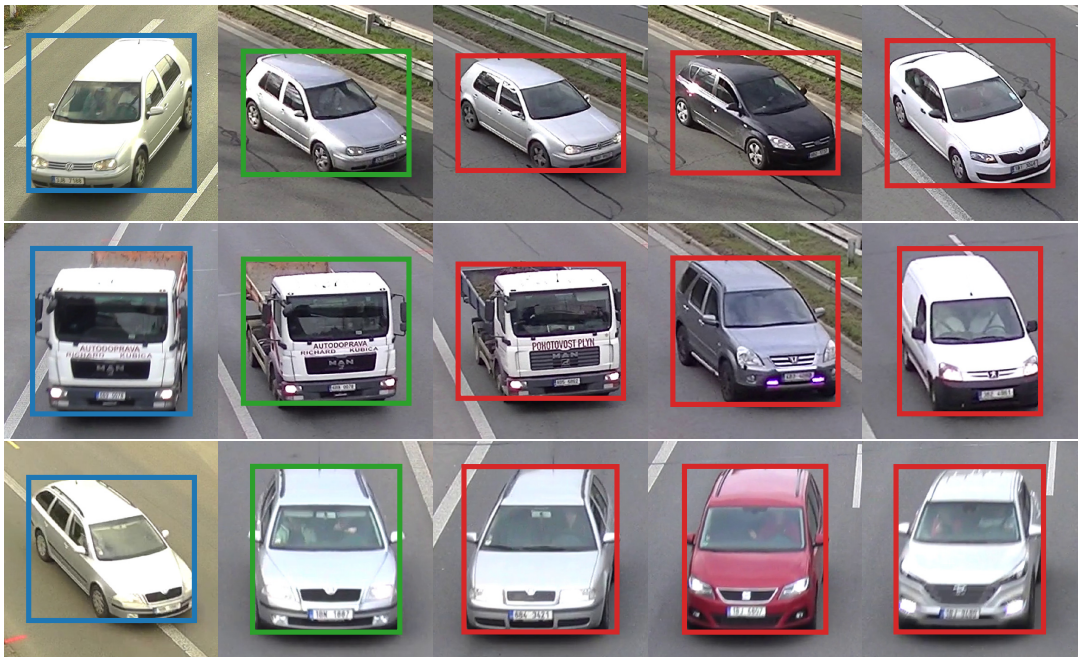

FIGURE 4.2: Examples of queries, positive, and negative samples. The negatives are sorted by difficulty from left to right (hard to easy) based on distances obtained from our re-identification feature vectors. It should be noted that the hardest negative sample has usually subtle differences (*e.g.* missing a small spoiler in the first row). Image source: [5].

Vehicle Re-Identification

Following the methodology from our previous paper [5], we first **fine-tuned the CNN** on vehicle *identification* task. We used ResNet-50 [74] and InceptionResNetV2 [173] with 2D detection/cropped images only, and the input image size was 331×331. The fine-tuning was done with Adam [174] optimizer, learning rate $1e-4$, and cross-entropy loss. We were not able to use our previously proposed modification using "unpacked" version of vehicle images [135] which is based on the construction of 3D bounding boxes as the input of the CNN due to limitations of viewpoints and already cropped images in *Track2*.

On the identification features we trained **LFTD network** [5] to aggregate the features in the temporal domain as there are multiple observations for the vehicle as they pass in front of the cameras. The LFTD network contains one fully connected layer with 1,024 output features and tanh non-linearity. Furthermore, the network contains a feature weighting mechanism which weights different elements of the feature vectors by different weights. The network is trained as a Siamese network.

It is possible to use a different distance function during LFTD network training. We used **Weighted Euclidean distance** which is expressed as

$$d_{\mathrm{WE}}(\mathbf{u},\mathbf{v}) = \sqrt{\sum_{i=1}^{D} w_i (u_i - v_i)^2}, \tag{4.1}$$

where $\mathbf{u}$, $\mathbf{v}$ are feature vectors and $\mathbf{w} = [w_1, w_2, \ldots, w_D]$ are learned weights.

We evaluated different variants of backbone networks together with the influence of using

image modifiers [113, 135] and pre-training the networks on different datasets. The complete results of our experiments are depicted in Table 4.3.

Vehicle Re-Id Design Changes Results of our submissions from the evaluation server showed unbalanced values compared to our evaluation (see Table 4.3). This led to designed changes in our methodology proposed above. Inspired by previous works [157, 175], we tried to replace our feature extractor by a much smaller CNN MobileNet [75] with feature vector dimensionality reduced to 128 dimensions. The second change was in replacing cross-entropy loss with triplet loss combined with **semi-hard batch sampling** [176]. The rest of our design remained the same. We tried multiple variants of *image modifiers* and pooling methods. The results can be found in Section 4.1.4.

Multi-Camera Tracking

Unlike the vehicle re-identification task, the multi-camera tracking task does not have to be solved by visual comparison of detected vehicles only. The problem can be solved even using **positional matching** with knowledge of GPS coordinates of cameras, known distances and time synchronization between them, and their calibrations. In this case, matching is based on a projection of the 2D point of vehicle trajectory from the image space into the world coordinate space (linear system in our case). These projections from multiple cameras can be matched with each other for every time step in order to obtain matching between tracks across multiple cameras.

It should be noted that the approach described below assumes that for each camera within the session, an overlap exists in the view area of the camera with at least one other camera in the same session.

This condition is satisfied for almost all cameras in test sessions, as can be seen in Figure 4.7. In other case, vehicles from a camera without any overlap cannot be matched with the rest of the cameras and the matching procedure had to be modified. However, with knowledge of time synchronization and distances between the cameras, this modification is straightforward.

FIGURE 4.3: *left:* Detections' bottom points used for localization of camera's view area. *right:* Corresponding points transformed to the linear space (viewpoint selected as the convex hull of these points).

Vehicle Trajectory Estimation Positional matching counts on an estimation of the trajectory points of each observed vehicle. The selection of a point from the vehicle detection may influence the precision of point localization in the world space. One solution is to construct the

3D bounding box [113, 135] around the vehicle and select the middle point of the vehicle base lying on the ground plane. However, this 3D bounding box construction is computationally expensive as it relies on the silhouette of the vehicle. We use the middle point of 2D detections' bottom-line provided instead, as this point performs the best from available data.

Transformation from Image to World Coordinate System The transformation process assumes that the calibration parameters for each camera are known. We used camera calibration provided with the dataset which was in the form of a homography matrix describing the transformation from the image plane to GPS coordinates in DD (*Decimal degrees*) format. The transformation between coordinate systems is a straightforward operation made only by matrix multiplication — homography matrix $\mathbf{H}$ multiplied by GPS coordinates in homogeneous format to transform from GPS to the image plane (*i.e.* inverted homography matrix multiplied by image point in homogeneous format to transform image plane point to GPS coordinates). Two cameras in the dataset (*c005* and *c035*) are fisheye cameras and thus compensation for the distortion of the point is necessary before transformation to GPS coordinate systems.

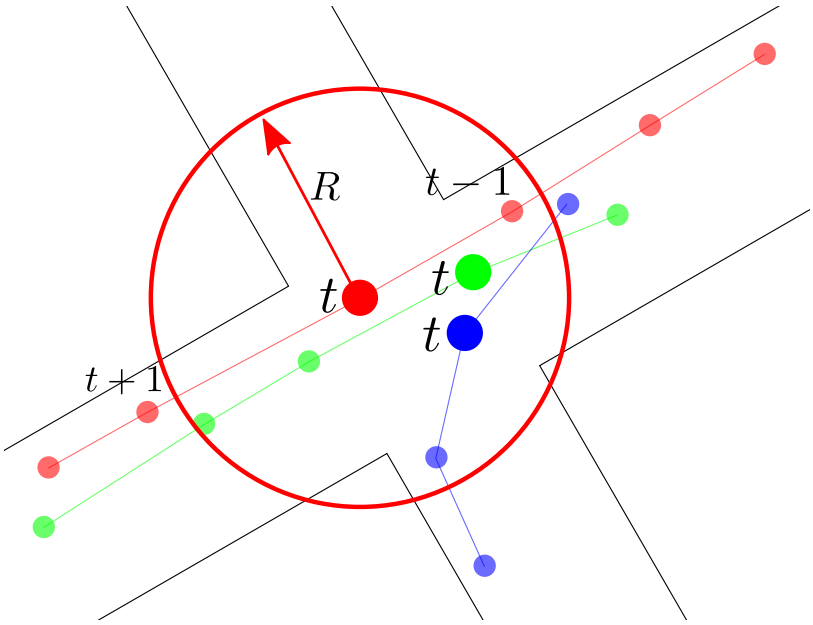

FIGURE 4.4: Visualization of the positional matching method. **Red** and **green** tracks corresponds to same vehicle observed by multiple cameras. **Blue** track represents another track of another vehicle. Positional matching iteratively determines if points from one trajectory correspond to points from another trajectory by constructing a circle with radius R in each time-step t and matches are accumulated to the *matching matrix*. This matrix contains a score for each possible combination of camera-track pairs.

Projection of GPS to Linear System Since GPS coordinates are known in the DD format and not as positions on the flat plane, the distances and positions do not correspond precisely to the real world because of the curvature of the Earth. Although distances in the DD format can be computed by *Haversine formula*, they can potentially suffer by some inaccuracies, and thus transformation to the linear space was done. We used transformation from *EPSG:4326* to *EPSG:26975* (corresponds to *North Iowa* where the dataset was collected).

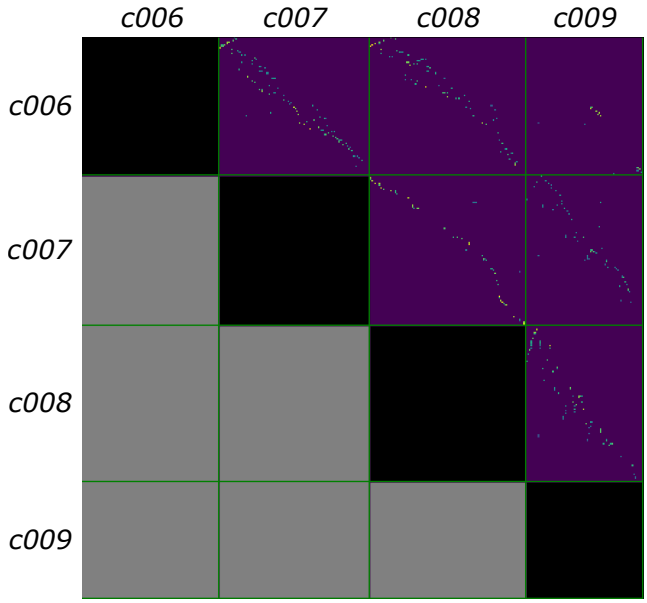

FIGURE 4.5: Matching matrix for *S02*. Each block size differs based on a count of tracks detected in single cameras.

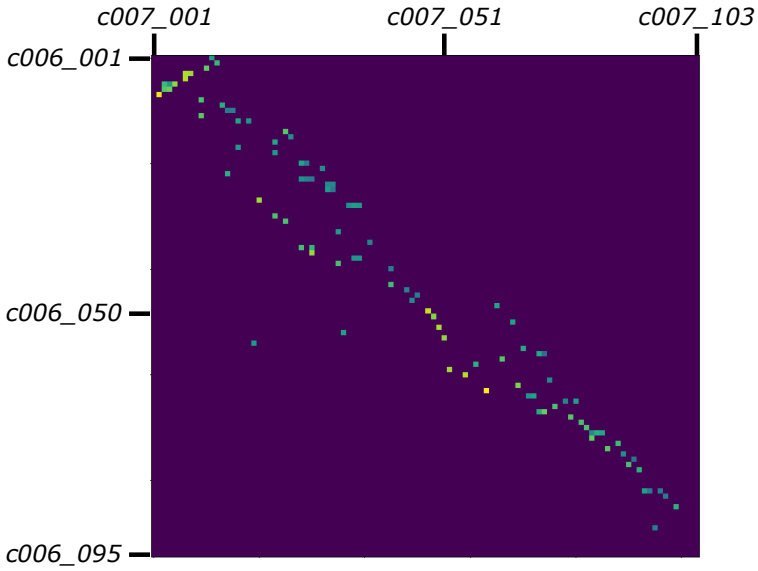

FIGURE 4.6: Matrix corresponding to sub-block *c006-c007* from Figure 4.5. Each cell contains the count of matches between the track in each row and track in each column.

Camera View Area Estimation A part of our solution is automatic detection of the camera view area (polygon covering part of the real world, where objects can be seen by a specific camera). For each of the observed vehicles, the two bottom corners of the vehicle's bounding box are transformed to the linear space. The convex hull of the points is used as a polygon covering the camera's view area. An example of the points used during the detection of a camera's viewpoint together with the corresponding linear space is depicted in Figure 4.3. Examples of localized viewpoints for a part of all cameras in a session can be seen in Figure 4.7. Our experiments show that it is convenient not to use all detections, but to limit these detections in some way — detections must be larger than 1000 pixels (in area), and all detections should

be no further than 300 meters from a camera in the linear space.

FIGURE 4.7: Localized viewpoints for part of all cameras in *S04* and *S05*.

Multi-Camera Tracks Positional Matching Positional matching between vehicle tracks observed by multiple cameras at one session is based on comparing mutual positions of individual trajectory points from multiple cameras in the real-world linear coordinate system in each time step. Trajectory points from each camera in a session are sorted by the time of their observation (*time steps*). For each time step and each trajectory point observed by one camera, we construct a circle in the linear space with radius R, and we are looking for trajectory points from other cameras in the session which are contained inside the constructed circle (for better understanding, please see Figure 4.4). These pairwise cameras–track *matches* are accumulated in a *matching matrix* M. This matrix contains all possible matches from each track in one camera to all tracks in the other cameras. An example of the matching matrix can be seen in Figure 4.5.

The matching matrix is split in pairwise camera blocks (see Figure 4.6). In each row of these blocks, we are looking for maximal accumulated values in other camera blocks separately using *Linear Sum Assingment* solver. These maximal values correspond to the best matching tracks between all cameras in the session. Best matches are further processed and joined into bigger groups if some element of *pairs, triplets, quadruplets,...* is missing in the other set which has at least one shared element.

Even visual features can be employed in the proposed method for solving multi-camera tracking problem. We are able to extract features (by using the same convolutional neural network with pooling as described in Section 4.1.3) from vehicle tracks given by camera-track indices of the matching matrix and to construct a pairwise distance matrix with the same shape as the matching matrix. This distance matrix is then used for the weighting of elements in the matching matrix.

4.1.4 Experiments

This section describes the experiments done while evaluating both challenge tracks.

Variant	Sess	IDF1	IDP	IDR	Prec	Rec
R=5m	02	0.0640	0.1240	0.0431	0.1560	0.0543
R=5m + feats	02	0.0664	0.1315	0.0444	0.1608	0.0543
R=5m	02,05	0.0480	0.0264	0.2623	0.0510	0.5064
R=10m	02,05	0.0340	0.0181	0.2865	0.0365	0.5792
R=10m + feats	02,05	0.0358	0.0190	0.3015	0.0365	0.5792

TABLE 4.1: Results for different variants of *positional matching* from AIC evaluation server. *R* represents a different radius of circle for positional matching. **Feats** represent using visual features extracted from tracks.

Rank	Team ID	Team Name	IDF Score
1	21	UWIPL	0.7059
2	49	DDashcam	0.6865
3	12	Traffic Brain	0.6653
4	53	Desire	0.6644
5	97	ANU AI city tracking and Re-ID team	0.6519
6	59	Zero_One	0.5987
7	36	DGRC	0.4924
8	107	IIAI-VOS	0.4504
9	104	Owlish	0.3369
10	52	CUNY-NPU	0.2850
11	48	BUPT-MCPRL	0.2846
12	115	KITMCT	0.2272
13	108	FirstBird	0.2183
14	7	iter1004	0.2149
15	60	i-TRACK	0.1752
16	87	DukBaeGi	0.1710
17	79	Alpha	0.1634
18	**64**	**GRAPH@FIT**	**0.0664**
19	43	VPUteam	0.0566
20	128	YXWM	0.0544
21	68	BUPT_MCPRL_MTMCT	0.0473
22	45	Insight DCU	0.0326

TABLE 4.2: Final ranking for multi-camera tracking part (Track 1) of NVIDIA AI City Challenge 2019.

FIGURE 4.8: Example result of our positional matching method with a projection of trajectory points into a linear coordinate system. Arrows depict transformed points from image space to real-world space in a specific time-step (displayed camera frame).

Evaluation of Vehicle Re-Identification

We employed our own version of evaluation on the **training** data in the same manner as the official evaluation is performed. We extract 1 000 images from training data which were used as our *query* images. This query set was used for the evaluation of networks performance on the training set.

Comparison of different variants of our trained networks and big differences between the performance on the training and the testing data can be seen in Table 4.3. Our original network design (CNN trained for the identification task with the cross-entropy loss and addition CNN for time-pooling — LFTD) was tested with a different combination of training data for both these tasks (feature extraction, time pooling). We trained our network the CarsReId74k [5] or we pre-trained the network on this dataset, and we fine-tune on AIC-ReID training data after that.

The results on our evaluation set were promising. However, after evaluating on the testing set, the results were very unsatisfactory. A big performance drop can be seen when training and testing evaluations are compared. This is probably caused by the size of the AIC-ReID dataset as the number of vehicles and their images included in the dataset is rather small.

We tried to replicate at least the baseline results provided by the authors of the challenge [175]. We trained MobileNet with a triplet loss function for feature embedding (128 dimensions) with semi-hard batch sampling. Again, results based on our evaluation procedure were promising, contrary to the final obtained results. However, the performance on the testing set is better. The final rank for this part (Track 1) can be found in Table 4.4.

Training setup A feature extractor for the CarsReId74k dataset was trained with LR 0.0001, Adam optimizer, and batch size 16 for 50 epochs while fine-tuning on the AIC-ReID dataset

						Train evaluation					Test evaluation (server)				
Net	**Loss**	**Pooling**	**Mods**	**Ext. Data**	**Pool. Data**	**mAP**	**H@1**	**H@5**	**H@10**	**H@20**	**mAP**	**H@1**	**H@5**	**H@10**	**H@20**
RN-50	Xent	avg	-	CR	-	0.306	0.213	0.377	0.484	0.641	-	-	-	-	-
RN-50	Xent	LFTD	-	CR	CR	0.331	0.238	0.403	0.522	0.667	-	-	-	-	-
RN-50	Xent	LFTD	-	CR	CR+AIC	0.786	0.711	0.881	0.933	0.97	-	-	-	-	-
IRN-v2	Xent	avg	-	CR	-	0.357	0.256	0.447	0.562	0.699	-	-	-	-	-
IRN-v2	Xent	LFTD	-	CR	CR	0.375	0.273	0.468	0.592	0.73	-	-	-	-	-
IRN-v2	Xent	LFTD	-	CR	CR+AIC	0.766	0.678	0.875	0.933	0.969	-	-	-	-	-
RN-50	Xent	avg	IM	CR	-	0.297	0.209	0.361	0.472	0.623	-	-	-	-	-
RN-50	Xent	LFTD	IM	CR	CR	0.311	0.219	0.381	0.492	0.645	-	-	-	-	-
RN-50	Xent	LFTD	IM	CR	CR+AIC	0.789	0.715	0.88	0.928	0.965	-	-	-	-	-
IRN-v2	Xent	avg	IM	CR	-	0.346	0.251	0.423	0.543	0.69	-	-	-	-	-
IRN-v2	Xent	LFTD	IM	CR	CR	0.362	0.259	0.451	0.577	0.723	0.0568	0.1141	0.1141	0.1179	0.1331
IRN-v2	Xent	LFTD	IM	CR	CR+AIC	0.766	0.683	0.87	0.925	0.968	-	-	-	-	-
RN-50	Xent	avg	-	CR+AIC	-	0.844	0.768	0.942	0.972	0.99	-	-	-	-	-
RN-50	Xent	LFTD	-	CR+AIC	CR	0.741	0.655	0.85	0.91	0.956	-	-	-	-	-
RN-50	Xent	LFTD	-	CR+AIC	CR+AIC	0.983	0.976	0.993	0.996	0.998	-	-	-	-	-
IRN-v2	Xent	avg	-	CR+AIC	-	0.988	0.981	0.997	0.999	1	-	-	-	-	-
IRN-v2	Xent	LFTD	-	CR+AIC	CR	0.978	0.968	0.99	0.994	0.996	0.2329	0.3536	0.3555	0.3650	0.4068
IRN-v2	Xent	LFTD	-	CR+AIC	CR+AIC	0.992	0.989	0.995	0.995	0.996	0.2420	0.3498	0.3508	0.3574	0.3926
RN-50	Xent	avg	IM	CR+AIC	-	0.829	0.752	0.928	0.965	0.988	-	-	-	-	-
RN-50	Xent	LFTD	IM	CR+AIC	CR	0.726	0.638	0.833	0.896	0.948	0.1428	0.2861	0.2871	0.2928	0.3137
RN-50	Xent	LFTD	IM	CR+AIC	CR+AIC	0.982	0.972	0.994	0.996	0.997	-	-	-	-	-
IRN-v2	Xent	avg	IM	CR+AIC	-	0.986	0.978	0.997	0.999	1	-	-	-	-	-
IRN-v2	Xent	LFTD	IM	CR+AIC	CR	0.976	0.963	0.992	0.997	0.998	0.2311	0.3622	0.3631	0.3641	0.3992
IRN-v2	Xent	LFTD	IM	CR+AIC	CR+AIC	0.991	0.986	0.996	0.999	1	0.2449	0.3707	0.3717	0.3755	0.4240
MobNet	Tri	avg	-	AIC	-	0.973	0.953	0.997	0.999	0.999	0.2883	0.3916	0.3916	0.4002	0.4496
MobNet	Tri	LFTD	-	AIC	AIC	0.976	0.959	0.995	0.998	1	0.2582	0.3432	0.3451	0.3489	0.3850
MobNet	Tri	avg	IM+Flip	AIC	-	0.962	0.934	0.995	0.998	0.999	-	-	-	-	-
MobNet	Tri	avg	Flip	AIC	-	0.989	0.978	1	1	1	**0.3157**	**0.4221**	**0.4221**	**0.4278**	**0.4270**

TABLE 4.3: Results for different variants of CNN feature extractors trained using different training setups (dataset used, network design, time pooling, data augmentation) and big gaps in our evaluation on training data and official evaluation. **Net**: RN-50 — ResNet50, IRN — InceptionResNet, MobNet — MobileNet; **Loss**: Xent — cross-entropy loss, Tri — Triplet loss; **Pooling**: avg — average time-pooling, LFTD — our time-pooling method [5]; **Mods** (data augmentation used while training): IM — Image modifications [113, 135], Flip — Horizontal flip of image; **Extractor/pooling data** (data used for training): CR — CarsReId74k, AIC data, CR+AIC combination of them.

was done for 20 epochs with the same hyperparameters. We use image modifications (IM) during training as proposed by Sochor *et al.* [113, 135] — specifically, we use **alterHSV** and **imageDrop**.

In the case of MobileNet with triplet loss trained for feature embedding, batch size 80 (4 samples for 20 vehicle identities) was used. The network was trained with LR 0.0003 with Adam optimizer for 150 epochs.

Feature aggregation network (LFTD) was trained for weighted euclidean distance (WE) with LR $10^{-4.4}$ using Adam optimizer, contrastive loss with margin 2.0, 30 rounds of hard negative mining and final features length 1024.

Evaluation of Multi-Camera Tracking

We tried to compute positional matching for different circle radius $R = \{5, 10\}$ with and without visual features. Generated files with results contained ≈ 3 millions of rows and the obtained results are unsatisfactory. This was probably caused by *joining* obtained matching set to bigger sets as this results in the corruption of time constraints. This led to the selection of a large number of false positive tracks which was confirmed by evaluation of session S02 only with better IDF1 score. Unfortunately, due to time reasons we were not able to process more experiments and our method still needs more evaluation. All evaluated variants can be found in Table 4.1. The final rank for this part (Track1) can be found in Table 4.2.

Rank	Team ID	Team Name	mAP Score	Rank	Team ID	Team Name	mAP Score
1	59	Zero_One	0.8554	43	80	IFP	0.3266
2	21	UWIPL	0.7917	44	1	SJSU_Anastasiu	0.3242
3	97	ANU AI city tracking and Re-ID team	0.7589	**45**	**64**	**GRAPH@FIT**	**0.3157**
4	4	expensiveGPUs	0.7560	46	104	Owlish	0.3090
5	12	Traffic Brain	0.7302	47	33	HRI-SH	0.3081
6	53	Desire	0.6793	48	50	AHUer	0.3047
7	131	XINGZHI	0.6091	49	76	GOGOGO	0.3039
8	5	UMD_RC	0.6078	50	79	Alpha	0.2965
9	78	MVM	0.5862	51	63	QMUL	0.2928
10	127	flyZJ	0.5827	52	6	UWACS	0.2912
11	92	APTX	0.5725	53	108	FirstBird	0.2867
12	154	XJTU-SMILES Lab	0.5693	54	46	SkyRoads	0.2766
13	27	INRIA STARS	0.5344	55	87	DukBaeGi	0.2763
14	107	IIAI-VOS	0.5229	56	120	YXX	0.2713
15	132	AlphaVehicle	0.5096	57	117	AI Pioneers	0.2693
16	114	Casia&Sg.panasonic&Bjtu	0.5040	58	145	Luo Jia Team	0.2599
17	23	KFC	0.5028	59	68	BUPT_MCPRL_MTMCT	0.2531
18	24	Avengers5	0.4998	60	43	VPUteam	0.2505
19	40	AI Bandits	0.4631	61	57	UTF-Puma	0.2481
20	48	BUPT-MCPRL	0.4610	62	55	reiddoneright	0.2451
21	7	iter1004	0.4406	63	18	Team Argus	0.2347
22	37	VCA	0.4195	64	62	CQUPT_EINI	0.2345
23	52	CUNY-NPU	0.4096	65	91	SJK	0.2228
24	14	CVHCI-KIT	0.4014	66	85	Bohemian Rhapsody	0.2184
25	113	HCMUS	0.4008	67	49	DDashcam	0.2176
26	70	helloketty	0.3960	68	25	GIST	0.2110
27	54	zhengge	0.3922	69	159	Walrus	0.2063
28	36	DGRC	0.3887	70	146	NCTUAI	0.2018
29	35	VD-blue	0.3814	71	163	TeamFellows	0.1748
30	41	SYSUITS	0.3769	72	139	Alpha_TSZ	0.1627
31	30	CheeseEgg	0.3741	73	125	BDTitan	0.1598
32	17	CSAI	0.3723	74	28	228Office	0.1583
33	51	ZJU	0.3689	75	15	ReId-this	0.1559
34	22	singlerace	0.3675	76	116	Conduent Labs India	0.0852
35	89	MMVG-AlibabaAIC-INF	0.3566	77	44	BUPT-CSD-Vision	0.0782
36	26	SYSU-ISENET	0.3503	78	58	Ann Arbor AI Amateurs	0.0340
37	124	BUPTCP	0.3496	79	45	Insight DCU	0.0322
38	96	SDU&Oeasy	0.3430	80	60	i-TRACK	0.0146
39	72	VehicleJian	0.3378	81	19	UCF_reid	0.0025
40	20	TJU0432	0.3339	82	128	Robint	0.0022
41	29	NCTU-NOL	0.3325	83	13	KAIST MSC	0.0004
42	47	ZJU_ReID	0.3317	84	133	AIIT-Jack	0.0003

TABLE 4.4: Final ranking for the re-identification part (Track 2) of NVIDIA AI City Challenge 2019.

4.1.5 Conclusions

We participated in two tasks of the NVIDIA AI City Challenge 2019: the vehicle re-identification task and the multi-camera tracking task. Our solution for vehicle re-identification is based on a convolutional neural network and time pooling of the feature vectors extracted from the observed vehicles. For the multi-camera tracking part, we propose a method for matching vehicle trajectory points in the real-world linear coordinate system space. This approach can be also combined with the extraction of feature vectors for all observed and pre-matched tracks.

4.2 AI City Challenge — Automated Retail Checkout

Information here is a copy of the paper [9], and thus results section differs from the information provided in this thesis. The order of teams changed after evaluation by organizers on *test-set-B* (which was not offered during development). We participated in the first year of this Track — our solution looked quite strange (removal of information), but unexpectedly we reached very good results (runner-up). Also, some other teams used a similar approach, which slightly surprised us.

4.2.1 Abstract

In this paper, we present a solution for automatic check-out in a retail store as a part of *AI City Challenge 2022*. We propose a novel approach that uses the “removal” of unwanted objects —

in this case, body parts of operating staff, which are localized and further removed from video by an image inpainting method. Afterward, a neural network detector can detect products with a decreased detection false positive rate. A part of our solution is also automatic detection of ROI (the place where products are shown to the system). We reached 0.4167 F1-Score with 0.3704 precision and 0.4762 recall which placed us at the *7th* place of *AI City Challenge 2022* in corresponding *Track 4*. The code is made public and available on GitHub[1].

4.2.2 Introduction

Self-service is a trend that is extending to more and more aspects of daily life (e.g. airport check-ins, automated teller machines at a bank, *etc.*). Customers are becoming more and more familiar with self-service systems. The triumphant procession of self-service systems seems to be extending to supermarkets. New automated self-checkout systems enable shoppers to scan, bag, and pay for their purchases without or with minimal help from store personnel. Retailers expect to reduce their costs and gain more flexibility by introducing self-checkout systems. One cashier can now serve multiple customers simultaneously to use staff time efficiently. Shorter checkout queues, a faster checkout process, more privacy, and greater control for the customers are the key arguments being used to convince the retailers to introduce the new self-checkout systems [177].

This year's *AI City Challenge 2022* [178] contains a new Track named *Multi-Class Product Counting & Recognition for Automated Retail Checkout*. This Track aims to automatically detect and report products present in front of a camera view and help during retail store checkout. The goal is to report all products — meaning product name (ID) and the time when the product was present. All products are present in the camera view in some defined area — this is a condition typically realizable in real-world scenarios. In this case, the defined area is a white tray which can be seen in Figure 4.9. The tray position is not defined and must be localized automatically (this process is described in Section 4.2.4).

Products may be occluded or very similar to each other, which may cause problems in detecting proper products and reporting their presence. One goal of the *AI City Challenge 2022* [178] is also to suppress an advantage of external data. Only provided or synthetic data can be used for the training of models (information about these data is available in Section 4.2.5). Our proposed solution comprises multiple sub-tasks that are done in the following order: person detection, image inpainting, ROI detection, detection of products, tracking of products, and tracks' post-processing. The individual steps and their benefits are described below in Section 4.2.4.

4.2.3 Related Work

In this part, a brief overview of related works from every field used during this challenge is provided. Relevant methods and models for automated retail store checkout (Section 4.2.3), multi-class object and body part detection (Sections 4.2.3, 4.2.3), multiple object tracking (Section 4.2.3) and image inpainting (Section 4.2.3).

[1] https://github.com/BUT-GRAPH-at-FIT/PersonGONE

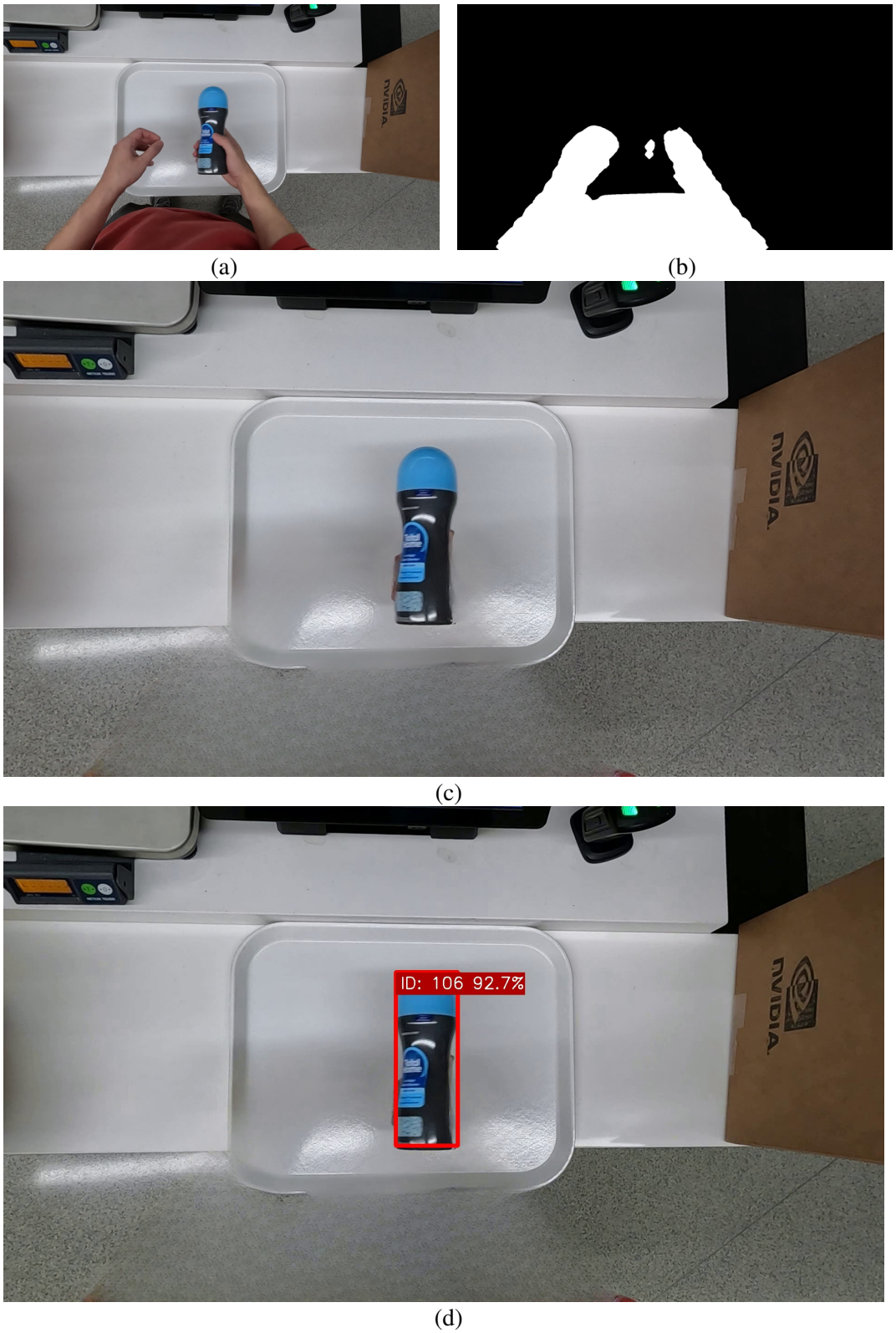

FIGURE 4.9: Our proposed solution for product detection. (a): Original video frame. (b): Detected mask of a person by CNN. (c): Removed "unwanted" parts by image inpainting algorithm. (d): Resulting detection of a product by CNN detector.

Automated Retail Store Checkout

Self-service checkout systems can be divided into two main categories — centralized or decentralized systems. Centralized systems are located at store exits, often created by self-checkout terminals or tunnel scanners. Decentralized systems use handheld scanners or mobile phones. In both cases, checkout depends on reading RFID [179, 180, 181, 182], EAN or QR code tags [180, 183] from retail items.

Automated store checkout might be extended from reading tags to recognizing items during checkout based on visual features of the item and its appearance in general. Aquilina and Saliba [184] presented a method for retail store automated checkout using SCARA robots. Their solution is based on a four-axis robotic system with machine vision, which automatically transfers items placed by a customer on a conveyor to the container, recognizes them, packs them, and prepares a total bill.

James *et al.* [185] proposes to use conventional multi-class detectors based on convolutional neural networks to detect and recognize items from a single RGB image.

Multi-class Object Detection

Convolutional Neural Networks dominate in object detection of their accuracy compared to older techniques [88, 186, 143, 187, 188, 87].

Single Shot Detectors (SSD) are one of the detection meta-architectures performing multi-class object detection. Liu *et al.* [88] published a study on a method called SSD which uses a single feed-forward CNN to directly predict classes without a second stage classification operation processing the proposed boxes. The term itself can denote the whole class of such detectors. Typical representatives of this group (aside from the original SSD detector) are also Multibox [143], YOLO-series detectors [186, 15, 14, 82, 18] or the Region Proposal Network (RPN) stage of the Faster R-CNN [13], which are used to predict class-independent box proposals.

In recent months, anchor-free detection models have taken the lead in this field. Most of them evolved from the anchor-based methods described in the previous paragraph. Chen *et al.* [188] revisits the concept of feature pyramids networks (FPN) for one-stage detectors in the *YOLOF* detector. The authors proposed a way to utilize only a one-level feature for detection instead of the divide-and-conquer optimization problem solution inside FPN.

Feng *et al.* [187] introduces *TOOD: Task-aligned One-stage Object Detection*. This method combines object localization and classification from attention maps into alignment metrics, which better balances learning task-interactive and task-specific features. Together with the proposed *Task Alignment Learning* for anchor position optimization, this step helps them surpass previous one-stage detectors.

YOLOX is an anchor-free evolution of YOLO series detector models created by Ge *et al.* [87]. They also adapt advanced detection techniques such as a decoupled head and the leading label assignment strategy SimOTA. YOLOX outperforms comparable models YOLOv3 [14], YOLOv4 [82] and YOLOv5 [18] by a large margin.

Multiple Object Online Tracking

Simple Online Real-time Tracker (SORT) from Bewley *et al.* [16] is a visual multiple object tracking framework that relies on fundamental data association and state estimation techniques based on Kalman filter [189]. It estimates objects' identities on-the-fly using detections from past and current frames only. It also supports object re-entry in a predefined time window and partial occlusion. The SORT tracker was extended using deep association metric based on image features as DeepSORT tracker [17].

ByteTrack tracker by Zhang *et al.* [190] is a method for multiple objects' tracking similar to DeepSORT. ByteTrack does not filter out low score detections (*e.g.* occluded objects, small objects) and associates almost every detection instead of the ones with a score over a threshold. In this case, the tracker is processing detections from the YOLOX detection model; similarity features for detection to track association are extracted from the FastReID model [191]. They outperform many available online trackers.

Detection of Body Parts

A helpful step in automated retail store checkout solutions might be detecting human body parts — especially hands. The known position of hands can be used as additional information when a human is holding individual items during checkout.

The 3D hand pose and shape estimation from a single RGB image has many real-world applications, such as robotics, augmented reality or gesture recognition. The goal is to localize a human hand's semantic keypoints (*i.e.* joints) in the 3D space. It is an essential technique for understanding human behavior and human-computer interaction. Various deep learning methods have been used. They can be divided into two main categories: joint estimation via keypoint detection and object detection with instance segmentation.

Hand-joint detection is based on keypoint regression networks primarily. This covers also SCNet [192], HRNet [193], or baseline methods of 3D hand pose datasets like FreiHand [194] or InterHand2.6M [195]. These models can be further enhanced by using Unbiased Data Processing (UDP) [196], during the training phase.

Another way to solve this task is employing *object detector with instance segmentation* — mask of the detected object. DetectorRS [97], PointRend [197], YOLACT [79] and HTC [198] are a few samples of such detectors. The knowledge of object masks might be used for image pre-processing to boost the performance of the aforementioned multi-class object detectors.

Image Inpainting

Image inpainting refers to the task of completing missing regions of an image. This fundamental computer vision task has many practical applications, such as object removal and manipulation, image retargetting, image compositing, and 3D photo effects.

Previously used *patch-based methods* (copy-pasting patches from known regions) or *diffusion-based methods* (color filling using partial differential equation) are outdated nowadays. Generative adversarial neural networks (GANs) are taking the lead in recent years, and they are still making great progress in generated image quality and preciseness. GAN models are composed of two main components, *generator*, the part responsible for image synthesis, and *discriminator* in the role of referee.

Two-stage networks predict an intermediate representation of an image in the form of edge, gradient, segmentation map, or a smoother image for final output enhancement. In order to augment the adversarial loss and suppress artifacts, many works often train the generator with additional reconstruction objectives such as perceptual, contextual, or l_1 loss.

Yu *et al.* [199] aimed at expanding the receptive field of the network by incorporating dilated convolutions to the generator and designing contextual attention to explicitly let the network borrow patch features at a global scale.

To enhance global prediction capacity, Zhao *et al.* [200] propose an encoder-decoder network that leverages style code modulation for global-level structure inpainting.

Suvorov *et al.* [26] propose to use Fourier convolution to acquire a global receptive field and segmentation networks to compute perceptual loss to achieve better performance. Furthermore, feature gating such as gated or partial convolution is proposed to handle invalid features inside the hole.

4.2.4 Methodology

As mentioned earlier, our proposed method for checkout consists of several steps. Each of the steps is described in more detail in the following sections.

Person Removal

As the data provided consists of synthetic images of individual products, these synthetic images were used for training by inserting them into ordinary images (more details are in Section 4.2.5). Objects are inserted into the "free space", and therefore, these products were often isolated in the frame during training, and there were no other objects near the annotated products. For this reason, it has often been the case during inference that the worker's hands or body are detected as products even in cases where no product occurs in the scene (as can be seen in Figure 4.10). Therefore, we decided to use an image inpainting method to remove the person, which significantly reduced the false positive detection rate.

The method used to "delete" a person is LaMa [26]. This method requires an image and a related mask as its input. Thus, it is necessary to detect the person's mask as the first step. Instance segmentation methods are used for this purpose — we tried several different methods for instance segmentation from *MMDetection* toolbox [201]. In particular, we tested the models *DetectoRS* [97], *HTC* [198], *PointRend* [197], and *YOLACT* [79]. All of these methods suffer from some inaccuracies. We combined the outputs of all the methods — some examples can be seen in Figure 4.11.

The usage of dilation additionally enhances the detected person masks. It helps to make some higher external borders, and in some failure cases, "ghost objects" do not appear in the frame. The dilation value must be set carefully as low dilation keeps the mentioned "ghost objects". On the other hand, too high a value removes potentially necessary information. The inpainting is processed frame-by-frame, so in future work, video inpainting could be used (*e.g.* [202, 203, 27]). However, these methods have high memory requirements, and for our goal, image inpainting is sufficient. An example of the convenience of the method used can be seen in Figure 4.12.

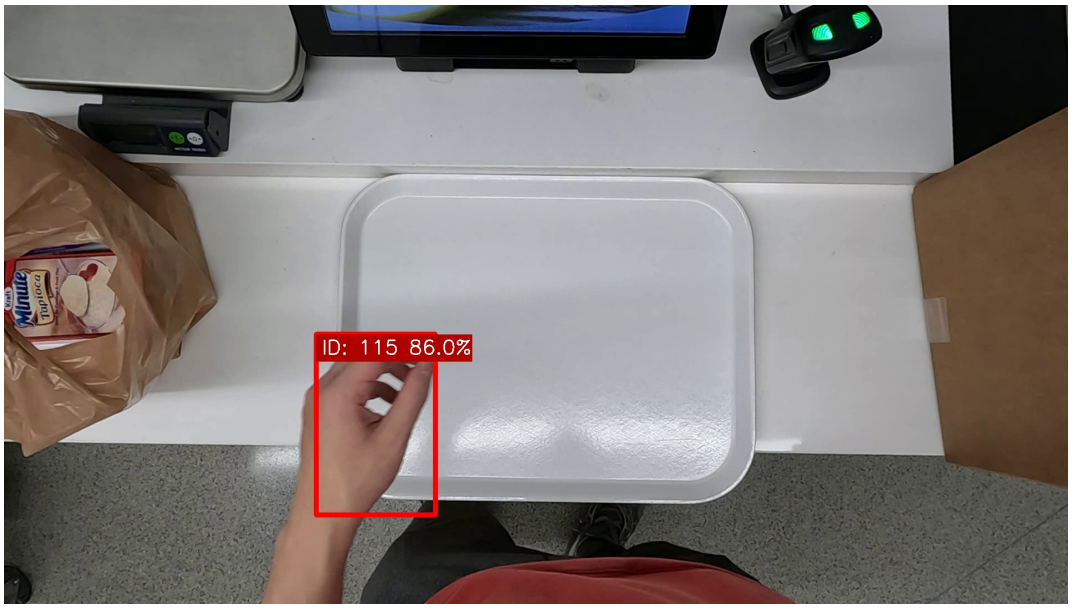

FIGURE 4.10: False positive product detection on person body in original frame.

ROI Localization

The trained model also had problems that it detect objects outside the required area, and at the same time, the goal of the track is to report objects above the "white tray". An example of unwanted detection outside the area can be seen in Figure 4.13. The localization of ROI (region of interest) is made automatically; the first step in this process is to extract the background image. This image for each scene is extracted by following: Gaussian Mixture Model [204] extracts background the part of each frame of the video sequence (with the usage of previous frames), and the mean value of all these background images is computed as the resulting background model. For the computation, inpainted video frames are used, as it makes the process easier by removing unnecessary objects (persons). Examples of resulting background images are depicted in Figure 4.14 *top row*.

When the mean background image is extracted for the scene, the image is transformed to grayscale, and the Scharr operator (an enhanced variant of the Sobel operator) for edge detection is applied. The Scharr operator is applied in x and y direction and combined together (resulting edge detection is in Figure 4.14 — *bottom left*). A *flood fill* algorithm is used on this image with detected edges, which searches the same/similar values as a seed (in our case, the seed is in the image center, but can be set arbitrarily). In this way, all pixels until edges are connected and marked as the "tray" in our case. Detected ROI can be seen in Figure 4.14 — *bottom right*. When ROI is detected, it serves for filtering detections outside (with some extension). The resulting bounding box is an axis-aligned rectangle of all pixels found by the flood fill algorithm.

Product Detection and Tracking

As mentioned earlier in the Introduction (Section 4.2.2) the next step after person removal and ROI detection is the detection of the products themselves. For detection, we use a multi-class *YOLOX* [87] detector, which reaches high precision on public datasets evaluation together

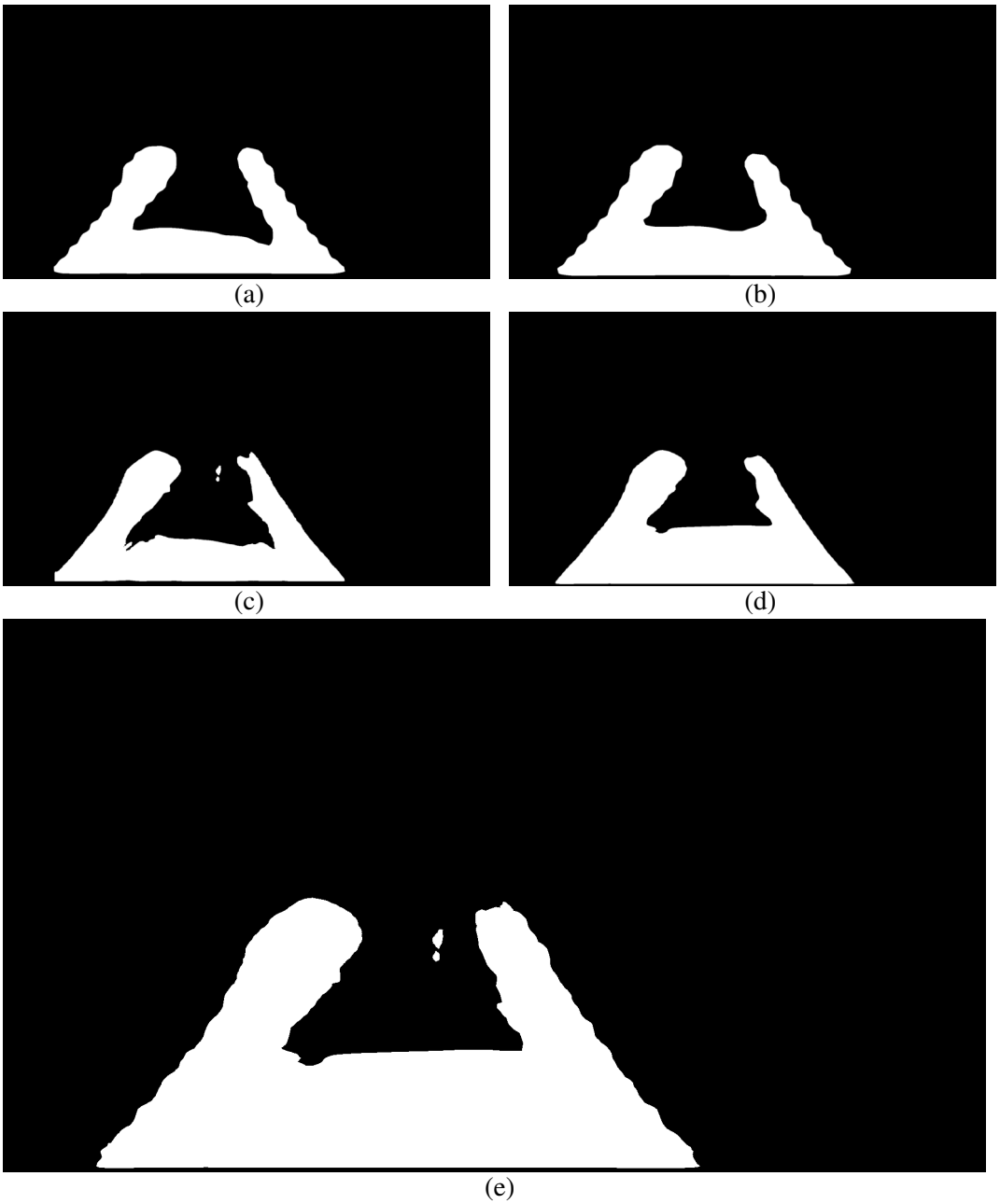

FIGURE 4.11: Construction of person mask for further "removal". (a): DetectoRS output. (b): HTC output. (c): PointRend output. (d): YOLACT output. (e): Combination of all the methods.

with fast inference speed. We trained our own model with 116 output classes on the provided dataset of generated objects as described in Section 4.2.5. We reached AP 97.47 % on the validation set during training. Together with object position, detection confidence, and class confidence are available, which are used further in the tracks post-processing. An example of object detection is available in Figure 4.15.

Together with the detection of products, tracking of individual products is performed. As a part of our solution, we tried two tracking algorithms — *SORT* [16] and *ByteTrack* [190]. Both algorithms work online with bounding rectangles of the detections and use the Kalman filter to predict the future positions and merge these detections/predictions to corresponding

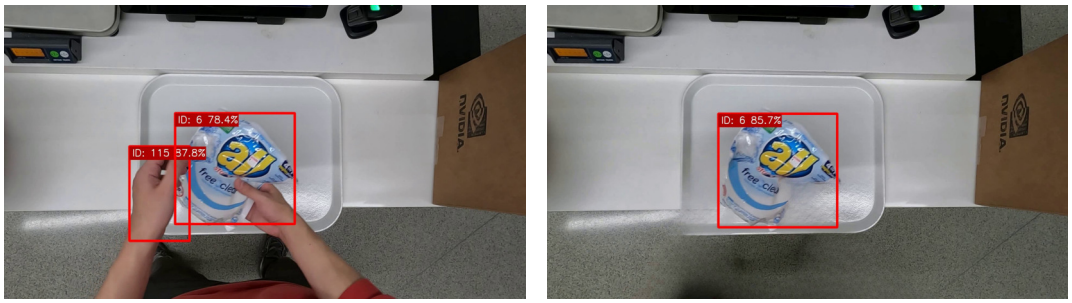

FIGURE 4.12: The difference between detection in the original and the inpainted frame.

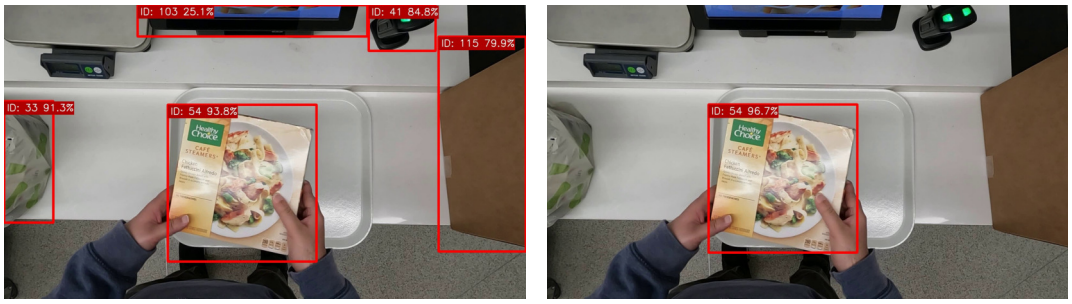

FIGURE 4.13: Detection of products without/with the usage of detected ROI.

tracks.

Tracks Post-processing

The last step in our pipeline is a post-processing of the detected tracks. Each track t^i contains detections d^i_j. Each detection d^i_j is composed of a bounding box, class ID, class confidence, and detection confidence as was mentioned previously. As a track can contain certain inaccurate class values (in one track t^i can be detections d^i_j with different class IDs) it is necessary to set a single class value for the whole track t^i.

For this purpose, classes are merged together by their class *id* as c^i_{id}. For each single class c^i_{id} in track t^i is computed its count $|c^i_{id}|$ and mean class confidence value $\overline{c^i_{id_{\text{class\_conf}}}}$ of all detections belonging to the corresponding class *id*. To each class *id* is computed its weighted confidence value as:

$$c^i_{id_{\text{conf}}} = \overline{c^i_{id_{\text{class\_conf}}}} \frac{|c^i_{id}|}{N^i},$$

where $N^i = |d^i_j|$ is the count of all detections in track t^i. The resulting class for the whole track t^i is the class with the highest weighted confidence value $c^i_{id_{\text{conf}}}$. In this way, not only a count of single class detections but also class confidences of each detection play a role in the decision of the final track class — this class is then reported as the final whole track class.

A part of the submission is also the time when the object was localized in front of the camera. The time should be any second (*integer*) when the object was above the ROI (detection described earlier in Section 4.2.4). As a part of our solution, we have frame numbers for each detection d^i_j. The class ID for each track is determined based on the procedure described earlier. Thus, the last necessary step is proper time computation. For each track, t^i is a computed list of time values in seconds as $d^i_{j_{\text{frame}}}/\text{fps}$ and rounded down (*math floor*) to the

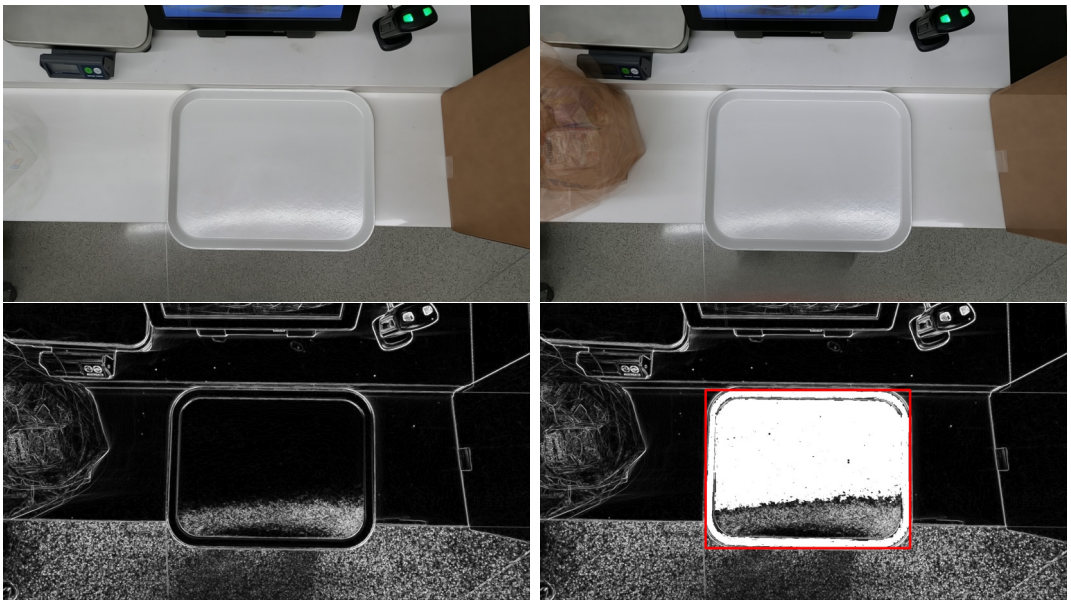

FIGURE 4.14: *Top row*: Mean background images for two sample scenes. *Bottom left*: Edges detection by the Scharr operator. *Bottom right*: Detection of the "tray" by the usage of flood fill algorithm; bounding rectangle in red.

nearest integer. These time values are accumulated into individual *bins* corresponding to proper values in seconds. The resulting (reported) second is the position of the bin with the highest accumulated value (the most detections in the proper second).

4.2.5 Experiments

This section describes the datasets used in this work, the performed experiments, and the achieved results. Datasets are divided into AIC Challenge Dataset for Track 4 and our generated synthetic dataset for the training of the aforementioned *YOLOX* detector.

AIC Challenge Track 4 Dataset

The dataset for multi-class product counting and recognition for automated retail checkout is provided as part of the *AI City Challenge 2022* (Track 4). This dataset contains 116,500 synthetic images, generated using a pipeline by Yao *et al.* [205] with masks for training, created from 116 different merchandise item models captured by a 3D scanner. Random background images, which are selected from Microsoft COCO [64], are used to increase the dataset diversity. Sample images, together with masks, for some classes, are shown in Figure 4.16.

The automated retail store checkout quality is evaluated on the testing part of the provided dataset. This part is formed from 25 recorded test videos. These videos capture different checkout procedures in a simulated environment from the top view. The task is to correctly identify and count items in the region of interest at different times. A sample of these data can be found in Figure 4.9 (a).

The test set is split between Test sets A and B with a ratio 20 % to 80 % accordingly. Test set A was published for testing and result evaluation over the AIC evaluation server. Test set B is dedicated for further evaluation and the final ranking.

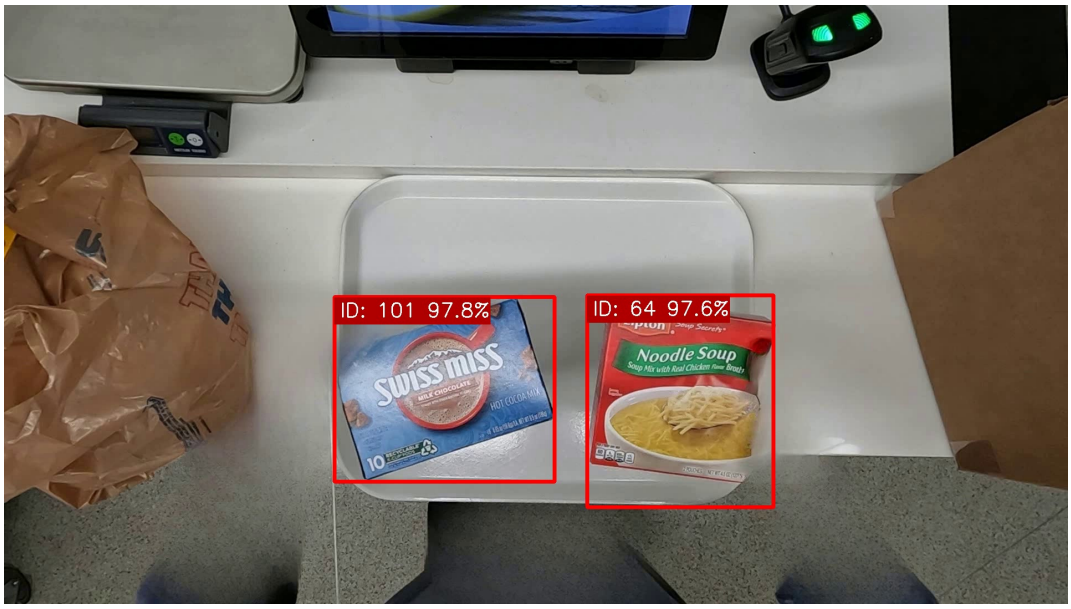

FIGURE 4.15: Multi-class detection with our trained *YOLOX* detector.

FIGURE 4.16: Sample of the synthetic dataset of products (*top row*) with generated object masks (*bottom row*).

Synthetic Dataset for Detector Training

The assignment of the *AI City Challenge 2022* is quite strict in the usage of external data, and thus we created our own dataset for the training of the detector as mentioned in Section 4.2.4. The only allowed "extra" data are those from other challenge tracks, and therefore we used these. We extracted 15,531 frames from Track 1 and Track 3 datasets and inserted synthetic images (Figure 4.16) into them. The inserted objects are cropped by the available masks; synthetic objects are thus freely placed in each frame (see Figure 4.17). In total, 100,000 and 20,000 images were generated for the training and validation set, respectively. For each (random) frame, 1 to 4 objects were randomly selected from the available synthetic dataset (Section 4.2.5) and randomly placed into the frame.

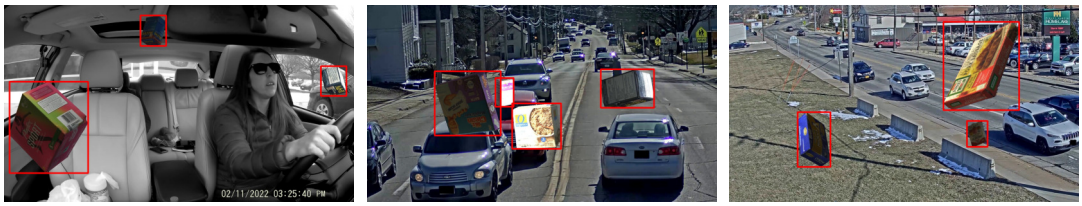

FIGURE 4.17: Sample images from our generated dataset for detector training with marked detections.

Implementation Details

As mentioned in Section 4.2.4, we need to localize an exact person mask by combination of different instance segmentation methods (*DetectoRS* [97], *HTC* [198], *PointRend* [197], and *YOLACT* [79]). All these methods are implemented in *MMDetection* toolbox [201] and also weights pretrained on *COCO* dataset [64] are available[2].

Once the instance masks are available, we use LaMa [26] for image inpainting — a model pretrained on *Places2* dataset [206] is also available[3]. Before the image inpainting application, the object mask is dilated. We experimented with different values and possible settings and found the best dilation solution with a cross kernel of size 9 and 3 iterations. The expansion of ROI for filtering is set to the value of 0.1 (expansion of width and height by 10 %).

We trained the *YOLOX* [87] detector on our generated dataset (Section 4.2.5); as the best, we found variant *YOLOX-L* trained for 75 epochs with input image size 640 × 640. All other training setting was equal to the original network setting (see website[4]). For tracking, we tried two trackers: *SORT* [16] and *ByteTrack* [190] with the period for which the track can be broken increased to 30 frames.

4.2.6 Evaluation

We tried several variants of the *YOLOX* network for object detection (Medium, Large, X-large) in two possible settings (network pretrained on COCO dataset and training from scratch). As a result, we selected variant *Large* trained from scratch. We also experimented with input image size — results can be seen in Tab. 4.5. All variants seem to be similar and produce very close results, and thus the input image size is probably not so important.

TABLE 4.5: Results with different input image size

Variant	F1-Score	Precision	Recall
640	0.4082	0.3571	0.4762
736	0.4000	0.3448	0.4762
800	0.4167	0.3704	0.4762

We also experimented with *SORT* and *ByteTrack* trackers. In our experiments, *ByteTrack* seems to be more stable and achieved the same result (0.4167 F1-Score) with lower image resolution (640 × 640) compared to the *SORT* tracker. However, all results are very close, and it is almost impossible to say which one is better. Both variants are implemented, and users can switch between them due to conditions. Our best result is 0.4167 F1-Score, so we placed *7th* in the public part of Track 4 challenge. The public leaderboard can be seen in Tab. 4.6.

Processing Data from Multiple Streams

An automated retail store checkout system should be installed at every checkout spot in the store to improve the Quality of Experience for customers. The solution based on computer vision and machine learning technologies could be computationally heavy and need adequate hardware for every checkout spot. Another possible approach is to use a distributed form

[2] https://github.com/open-mmlab/mmdetection
[3] https://github.com/saic-mdal/lama
[4] https://github.com/Megvii-BaseDetection/YOLOX

TABLE 4.6: Public leaderboard of *AI City Challenge 2022* Track 4

Rank	Team Name	Score
1	BUPT-MCPRL2	1.0000
2	SKKU Automation Lab	0.4783
3	The Nabeelians	0.4545
4	mizzou	0.4400
5	RongRongXue	0.4314
6	Starwar	0.4231
7	GRAPH@FIT	0.4167
8	HCMIU-CVIP	0.4082
9	CyberCore-Track4	0.4000
10	UTE-AI	0.4000

of processing using cloud-native applications. In this case, image data are transferred to the cloud, where each part can be computed individually using multiple workers. The result of each sub-tasks is passed to the following processing step. This approach corresponds with NetApp for distributed processing of the *5G Enhanced Robot Autonomy* project. The system proposed in this work is an excellent example of a task for distributed processing, where a single computational cluster located in the store or cloud may process many checkout spots.

4.2.7 Conclusion

We participated in *AI City Challenge 2022*, Track 4 called *Multi-Class Product Counting & Recognition for Automated Retail Checkout*. We proposed a novel approach based on image inpainting, which significantly improves the detection results and reduces the rate of false positive detections. As a part of our solution, we also automatically detect the region of interest and automatically segment out parts of humans and further "delete" them from the scene. We achieve competitive results with most other teams with the YOLOX-L detection network, which can run in real-time and trackers based only on bounding boxes (without deep learning). In the final leaderboard, we placed *7th* with F1-score 0.4167, which placed us in the first half of the participants.

4.3 Workshop on Maritime Computer Vision (MaCVi) — WACV

The text here is our part of the resulting paper [10] from the challenge. We participated in the object detection task. The goal was to detect different objects (boats, swimmers, *etc.*) in the sea from a drone view above the sea. Some example of how the data can look is depicted in Figure 4.18. The workshop is focused on security components — in case of some anomaly incident, drones can be at the place much faster than boats/ships and can help to localize people much faster.

We participated at **1st Workshop on Maritime Computer Vision (MaCVi)** in task *Object Detection v2*. The task was to detect objects in sea drones images. All our experiments were done on a personal computer with the following setup:

- System: Ubuntu 20.04.5
- CPU: Intel Core i7-11700K

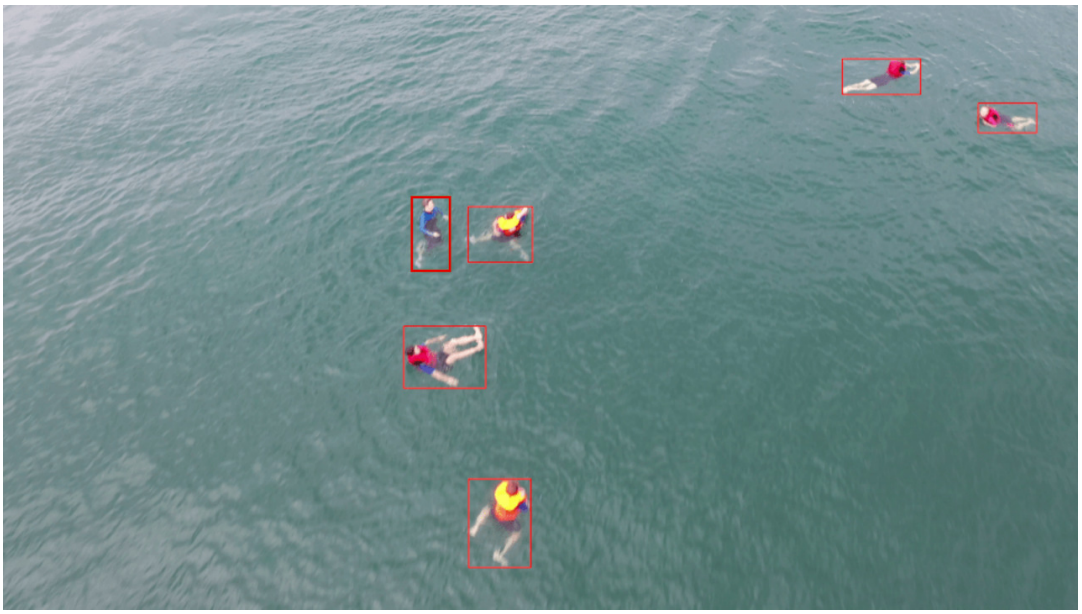

FIGURE 4.18: Sample image from MaCVi Workshop object detection task

- GPU: Nvidia RTX3090
- RAM: 128 GB

We experimented with 4 different methods and achieved results as described in Table 4.7.

Method	AP	AP_{50}	AP_{75}	AR_1	AR_{10}
YOLOv7	0.5169	0.8013	0.5511	0.4214	0.5796
TOOD	0.4707	0.7324	0.4919	0.3961	0.5361
Pix2seq	0.4194	0.7485	0.4056	0.3940	0.5379
DETR	0.3502	0.6763	0.3344	0.3205	0.4387

TABLE 4.7: Results of tested methods

In all cases, we used models pre-trained on *COCO* dataset and fine-tuned them on *SeaDronesSee* dataset provided by the challenge authors. If not mentioned, all training parameters were the same as in training scripts provided in relevant repositories. Only *YOLOv7* was able to run about $\approx$ 10 FPS; all other methods run about $\approx$ 1.5 FPS.

YOLOv7

We used *YOLOv7* repository [84] with prepared fine-tuning scripts — our variant was *YOLOv7-E6* with image size 1280 $\times$ 1280, batch size 4, and training for 300 epochs.

TOOD

Another tested method was *TOOD* [207]. We used implementation provided in *MMDetection* toolbox [208] — our variant was ResNet101 backbone with DCNv2 (*R-101-dcnv2*). The model was learned for 24 epochs with batch size 6.

Pix2seq

The next tested method was *Pix2seq* [12] which also provides prepared scripts for fine-tuning — we used ResNet50 backbone with image size 1333 $\times$ 1333. The model was trained for 40 epochs with batch size 4.

DETR

We also tried *DETR* [11] model. Similarly to *TOOD* we used implementation provided in *MMDetection* toolbox. The backbone was ResNet50, and the model was trained for 80 epochs with batch size 6.

Observations

We tried to fine-tune two "classical" CNN models (*YOLOv7*, *TOOD*) and also two models based on transformers (*Pix2seq*, *DETR*) on *SeaDronesSee* dataset. Our main observation is that "classical" CNN models still reach comparable results, and transformers do not provide significant result improvement.

Chapter 5

Horizon Estimation by Observation of Moving Objects

The text here is a copy of my first published paper [4]. The idea was to estimate the horizon and further make camera calibration based on some geometrical information present in a scene. Horizon contains two vanishing points, and thus localization of all 3 vanishing points is thus easier to solve. But this approach did not look promising, and I gave it up and turned my research to "landmarks" and ITS direction.

5.1 Abstract

This paper deals with the automatic estimation of the horizon in videos from fixed surveillance cameras. The proposed algorithm is fully automatic in the sense that no user input is needed per-camera, and it works with various scenes (indoor, outdoor, traffic, pedestrian, livestock, etc.). The algorithm detects moving objects, tracks them in time, assesses some of their geometric properties related to the object dimensions, and infers observations related to the position of the horizon. We collected a dataset of 47 public camera streams observing suitable scenes of various natures. We annotated ground truth horizons based on geometric properties in the images and by direct human input. We evaluate the proposed algorithm and compare it to human guesses – it turns out that the algorithm is on par with humans or it outperforms them in difficult scenes.

5.2 Introduction

Estimation of viewpoint is critical for understanding a given scene. Quick (and possibly not quite accurate) guess of the viewpoint is an important component of the *gist* of the scene [209, 210]. Creating such a representation of an image can be beneficial not only for human visual processing but also for computer vision. One of the most mentioned viewpoint aspects is the image's *horizon* [209]. Although horizon is a fairly intuitive characteristic of the viewpoint, in some complicated scenes (urban, occluded, ...), establishing the exact horizon can be complicated for humans and even more so for machines.

Horizon is very often simply explained as "the line where the sky meets the ground". This explanation corresponds to the intuitive human feeling about the horizon. The astronomical horizon is defined by the horizon plane, a plane that is perpendicular to gravity and located at the same altitude as the camera [209]. In a given image, the plane is only visible as a horizontal line, it is not dependent on the slant of the ground surface nor on the presence of occluders. Literature also defines a "horizon of an arbitrary plane" which is the line in the image where all parallel lines within this plane meet (i.e. the projection of the plane's ideal line) [211]. Horizontal lines that are parallel in the real world meet at a vanishing point that lies on the horizon. Horizon is thus naturally, and at no additional cost, established by algorithms that recognize vanishing points in Manhattan worlds, *e.g.* [212].

Horizon is very often used for camera calibration because two horizontal vanishing points lie on the horizon line. With the knowledge of the horizon line and the third (vertical) vanishing point, camera calibration can be done [213, 214]. An approach that estimates the vanishing points by connecting corresponding points of the same object and then constructs the horizon is often used for camera calibration with human tracking [41, 42, 43, 44, 45], where pedestrian's head and feet are connected by lines as the person moves across the scene and their intersection lies on the horizon line. Similarly, vanishing points and the horizon line can be localized for example by detecting pedestrians' toes in the ground plane [215]. Although camera calibration methods that avoid using horizon exist, they typically have some constraints, for example, known pedestrian height distribution [216, 217], 'Manhattan world' scene [218, 219, 220], or dominant motion only in one coherent direction [56, 221]. Horizon can also be used for estimating 3D scene geometry and object detection support [222, 223].

Our goal is to detect the horizon (in the sense of the ideal line of the ground plane perpendicular to gravity) in a single static (often surveillance) uncalibrated camera stream based on the motion of objects in the scene without any a priori constraints except that the majority of motions happen in horizontal planes (which share the same horizon by definition). Such a method can be later used for automatic camera calibration, scene understanding, and other computer vision tasks. We assume that the scene contains arbitrary objects (pedestrians, cars, dogs, cattle, machinery, ...) with an arbitrary height/size distribution. The scene does not need to be Manhattanian, and the motion can appear in any direction and in virtually any place in the scene. Existing methods for horizon estimation [224, 225] use a single static image without the assumption of a movement in the scene. Although motion can be used for horizon estimation for example in the form of cloud motion together with wind velocity [226], in our work we assume surveillance cameras without any constraints, no clouds thus need to be present in the scene, and no additional information is provided.

Although datasets with horizon position in image exist (HLW [227], ECD [228], YUD [229]), they only contain static images without any motion in the scene. To evaluate the method and to allow for future comparison (to our knowledge, there is no existing dataset dealing with this issue), we collected a dataset based on publicly available IP cameras. We manually constructed a ground truth by using geometric properties of objects in the scene, and we also collected human annotations which cast a light on the algorithm's performance and allow for its comparison to human (well trained and routinely used) *gist* of the scene mentioned earlier. The second contribution of this paper – after proposing the fully automatic horizon estimation algorithm – is making this dataset public[1].

[1] https://medusa.fit.vutbr.cz/trajectories

5.3 Horizon Estimation by Observing Motion

This section describes the use of object trajectories in the scene and it proposes an algorithm for fully automatic horizon estimation from them.

5.3.1 Trajectories and Horizons

Our horizon estimation is based on trajectories of objects tracked in a video (Figure 5.1). The objects can be arbitrary and heterogeneous in the scene (unlike many approaches we do not focus on a given known object class such as pedestrians or vehicles). The video is first transformed into a set of trajectories T by tracking foreground objects:

$$T = \{\mathbf{t}_1, \ldots, \mathbf{t}_N\} \tag{5.1}$$

Every trajectory is composed of individual object observations

$$\mathbf{t} = \{\mathbf{o}_1, \ldots, \mathbf{o}_{M_t}\} \tag{5.2}$$

which will be referenced as $\mathbf{t}(1), \mathbf{t}(2), \ldots, \mathbf{t}(M_t)$ for brevity. Each observation is described by a pair $(\mathbf{p}, \mathbf{d})$ whose parts are:

$$\mathbf{t}(i)^{\mathbf{p}} \quad - \quad \text{position of the observation} \tag{5.3}$$

$$\mathbf{t}(i)^{\mathbf{d}} \quad - \quad \text{dimensions of the observation} \tag{5.4}$$

Position $\mathbf{t}(i)^{\mathbf{p}}$ is naturally the 2D position of the center of the observation in the image. The dimensions $\mathbf{t}(i)^{\mathbf{d}}$ are possible dimensions of the observation (naturally width and height, possibly diagonals, other chord lengths, a measure related to the area, ...) and so $\mathbf{d}$ is potentially k-dimensional. Since we are dealing with quite low-resolution videos, we are only using the width, height, and diagonal of the axis-aligned bounding box in our experiments and k is thus 3 ($\mathbf{d}$ is three-dimensional). Here and later in the text, we are assuming 'normal' surveillance camera orientation; should the camera be placed so that the horizon turns 'vertical', such a situation would be easily detected and the algorithm can be adjusted in a straightforward way. We are omitting such adjustments here for simplicity.

Let us consider an arbitrary horizon

$$\mathbf{h} = (\mathbf{h}_m, \mathbf{h}_b), \tag{5.5}$$

a straight line in the slope-intercept form

$$y = \mathbf{h}_m x + \mathbf{h}_b. \tag{5.6}$$

We define the *distance* of a given observation with the position $\mathbf{t}(i)^{\mathbf{p}}$ from the given horizon $\mathbf{h}$ as

$$\Delta(\mathbf{t}(i), \mathbf{h}) = \left| \frac{\mathbf{h}_m \mathbf{t}(i)^{\mathbf{p}}_x - \mathbf{t}(i)^{\mathbf{p}}_y + \mathbf{h}_b}{\sqrt{\mathbf{h}_m^2 + 1}} \right| \tag{5.7}$$

If the object captured by a trajectory $\mathbf{t}$ is moving within the plane corresponding to the horizon (and no occlusion is involved etc.) and the dimensions $\mathbf{d}$ are correctly selected, the measured dimensions $\mathbf{t}(i)^{\mathbf{d}}$ must be linearly dependent on the distances $\Delta(\mathbf{t}(i), \mathbf{h})$, as illustrated in

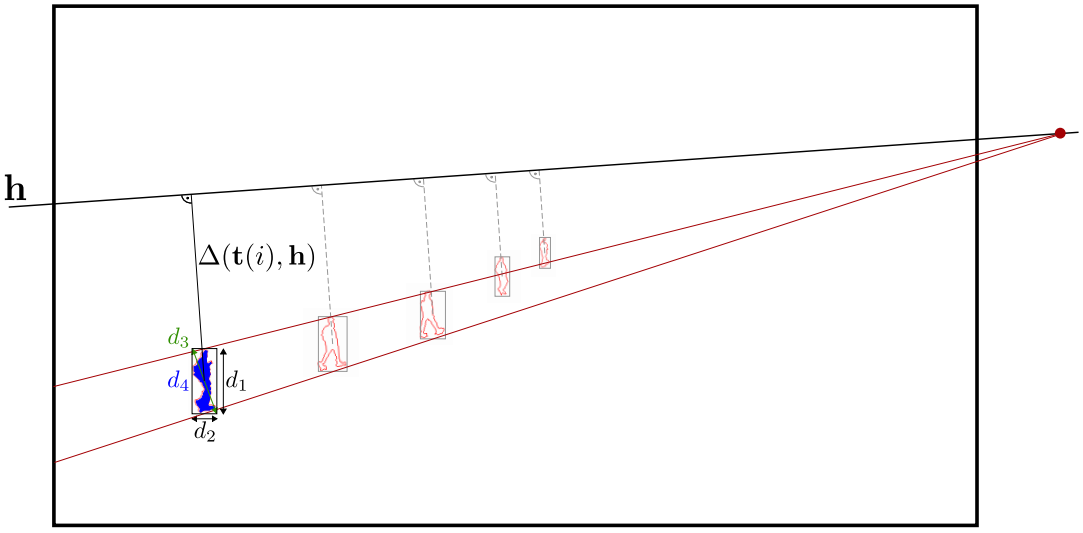

FIGURE 5.1: One possible trajectory (a few selected observations) in the video frame. Each observation consists of position $\mathbf{t}(i)^{\mathbf{p}}$ and possible dimensions $\mathbf{t}(i)^{\mathbf{d}}$. The distance $\Delta(\mathbf{t}(i), \mathbf{h})$ from horizon $\mathbf{h}$ is also depicted.

Figure 5.1:

$$\lambda \, \Delta(\mathbf{t}(i), \mathbf{h}) = \mathbf{t}(i)^{\mathbf{d}}. \tag{5.8}$$

5.3.2 Trajectory Contribution

Some scenes are typically problematic for horizon estimation; mainly scenes with one dominant direction of motion present. In such cases, the motion provides information about one vanishing point (situated on the horizon line), but not enough information about the whole horizon. It is thus necessary to compute the *contribution value* of the given track, where tracks in rare directions should contribute more since they provide valuable information. An example of a scene with one dominant motion direction is shown in Figure 5.2.

For computing the contribution value for each trajectory $\mathbf{t}$, histogram $\mathcal{D}$ is accumulated, which stores information about the directions of motions that appear in the scene. Histogram $\mathcal{D}$ contains B bins separating the interval of angles $(0, \pi)$ uniformly. The trajectory is divided to C parts uniformly, each part described by its motion direction (see Figure 5.3).

For every trajectory part vector defined by its endpoint indices a, b, i.e. $\vec{v} = \mathbf{t}(b)^{\mathbf{p}} - \mathbf{t}(a)^{\mathbf{p}}$, its angle to an arbitrary fixed reference horizon vector $\vec{\mathbf{h}}_{ref}$ (typically the vector horizontal in the image) is computed as

$$\phi = \arccos(\vec{\mathbf{h}}_{ref} \cdot \vec{v}), \tag{5.9}$$

with the assumption of normalized vectors. Angle ϕ determines which bin of histogram $\mathcal{D}$ is incremented for each trajectory's part. Histogram bin values are computed for all C parts of the trajectory and the final contribution value $\gamma(\mathbf{t})$ is computed by normalizing the sum of values of the corresponding histogram bins using the sum of all histogram values. It should be noted that for the efficiency of the computation, the histogram accumulation and evaluation of the contribution value is done on the fly with further trajectories coming. The histogram is first initialized by a non-zero value Γ configuring the initial estimation of the contribution values.

FIGURE 5.2: Example of a scene with one dominant direction of objects' motion. There are two directions of car movement on the road and a bidirectional stream of pedestrians on the sidewalk. Most of the tracks only testify to one vanishing point; only rare pedestrians crossing the road bring the missing information.

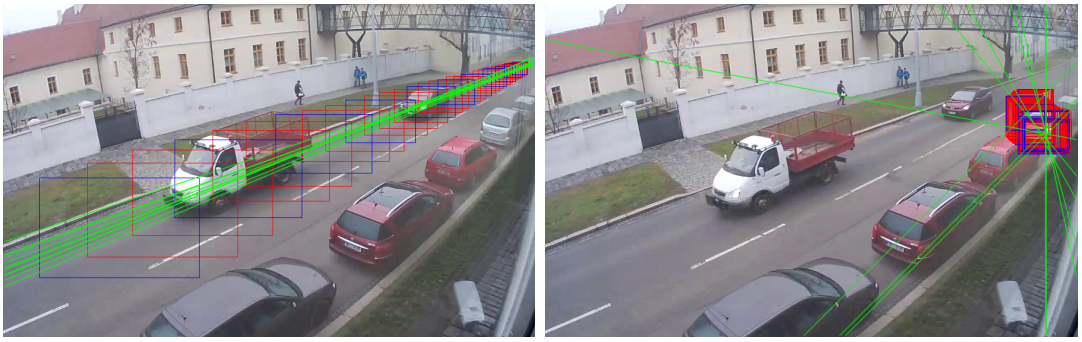

FIGURE 5.3: An example of computing the trajectory contribution value. Rectangles are observations, and endpoint observations of the trajectory parts are blue. Direction vectors $\vec{v} = \mathbf{t}(b)^{\mathbf{p}} - \mathbf{t}(a)^{\mathbf{p}}$ for each consequent couple of endpoint indices a, b are shown as green lines. **left:** Typical movement in the single dominant direction (car on the road). **right:** Motion in an uncommon direction (car leaving parking place).

5.3.3 Automatic Horizon Estimation

Parametric space $\mathcal{M} \times \mathcal{Y}$ of potential horizons is generated for horizon estimation. This parametric space $\mathcal{M} \times \mathcal{Y}$ samples the set of potential horizons $\mathbf{h} = (\mathbf{h}_m, \mathbf{h}_b)$. Each horizon is the line with slope $\mathbf{h}_m$, whose $\mathbf{h}_b$ is computed so that the line intersects the vertical central line of the image in the y coordinate. $\mathcal{M}$ are slopes of angles from interval $(-\beta, \beta)$ sampled with step s_m, and $\mathcal{Y}$ are vertical positions of the horizons from interval $(-y_{gen} f_h, y_{gen} f_h)$ sampled with step s_y, where f_h is the height of the video frame and y_{gen} is a configuration constant, in the experiments $y_{gen} = 80\,\%$, as illustrated by Figure 5.4.

For every potential horizon $\mathbf{h}$ and every trajectory $\mathbf{t}$, a confidence value is computed indicating

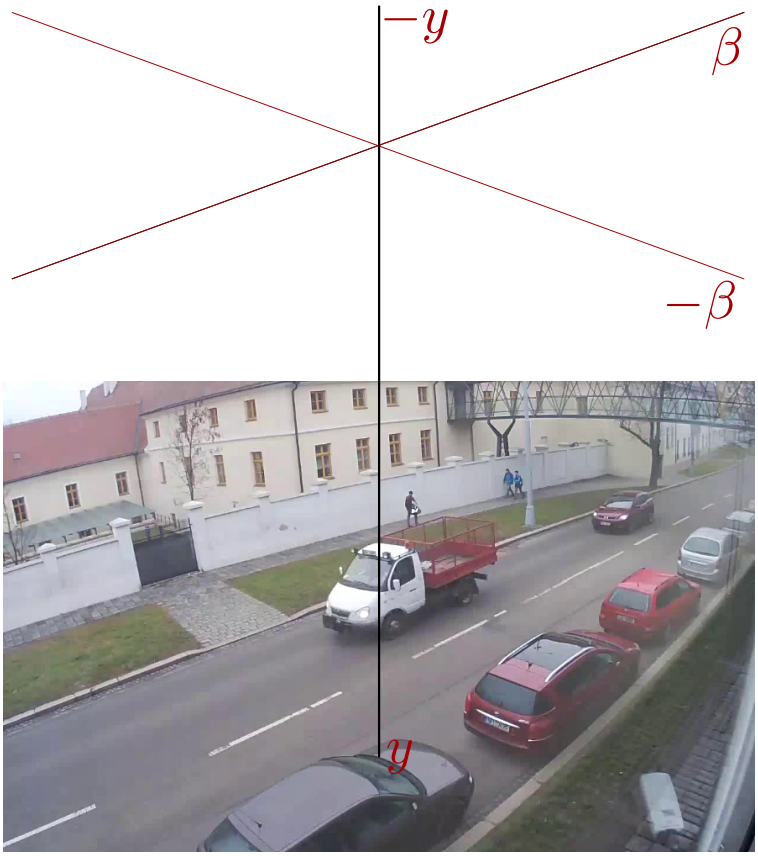

FIGURE 5.4: Generated potential horizons which are used for horizon estimation. In this case with value $y_{gen} = 80\,\%$ (y values from interval defined by 0.8 times the height of frame above and below top frame border) and slope angle values from interval $(-\beta, \beta)$.

how much the linear dependency assumption (5.8) of observation's distances from the horizon and their dimensions is met. In particular, least-squares linear regression is computed between the presumably linearly dependent values, and its error $\epsilon(\mathbf{t},\mathbf{h})$ is computed as the root mean square error (RMSE):

$$\epsilon(\mathbf{t},\mathbf{h}) = \sum_{j=1}^{k} \mathbf{w}_j \sqrt{\frac{1}{M_t} \sum_{i=1}^{M_t} \left(\mathbf{t}(i)_j^{\mathbf{d}} - \lambda\, \Delta(\mathbf{t}(i),\mathbf{h}) \right)^2}, \tag{5.10}$$

where $\mathbf{w}$ is k-dimensional vector of weights for individual dimensions of observation $\mathbf{t}(i)^{\mathbf{d}}$. Regression error $\epsilon(\mathbf{t},\mathbf{h})$ describes how (un)likely the potential horizon $\mathbf{h}$ can be the real horizon of trajectory $\mathbf{t}$. The trajectory's contribution value $\gamma(\mathbf{t})$ (Section 5.3.2) is also accounted for in the resulting confidence $c(\mathbf{t},\mathbf{h})$ of the trajectory $\mathbf{t}$ being assigned to potential horizon $\mathbf{h}$. The resulting confidence is then

$$c(\mathbf{t},\mathbf{h}) = \frac{1}{\delta_e + \epsilon(\mathbf{t},\mathbf{h})} \cdot \frac{1}{\delta_c + \gamma(\mathbf{t})}, \tag{5.11}$$

where parameters δ_e and δ_c control the weight of regression error and trajectory contribution value in the resulting confidence and they also ensure computational stability. This confidence is computed for every trajectory $\mathbf{t}$ and every potential horizon $\mathbf{h}$, and it is accumulated to the parametric space of potential horizons $\mathcal{M} \times \mathcal{Y}$ in a manner similar to the Hough transform and also following the work of Litman *et al.* [230] and the work by Zhai *et al.* [224]. After accumulating all the tracks' confidence, the horizon in the parametric space with the highest confidence is selected as the most likely solution.

5.3.4 Motion with One Dominant Direction

Although the trajectory contribution value, as described in Section 5.3.2, is computed to emphasize trajectories with unusual directions, in some specific scenes, the motion happens solely in one dominant direction. In such cases, the trajectories do not provide enough information about the horizon, only information about a single vanishing point corresponding to the dominant direction. This is typical mainly for traffic surveillance scenes, where only the single motion direction parallel to the road direction can be seen. An example of such a scene is shown in Figure 5.5 (a).

In such problematic scenes, there is a typical pattern in the parametric space $\mathcal{M} \times \mathcal{Y}$, where confidences are accumulated mainly for potential horizons all of which are coincident with the vanishing point given by the motion in the scene. The parametric space for such a scene is shown in Figure 5.5 (b). It manifests as a pattern resembling a 'line' in the parametric space – all potential horizons with high confidence intersect near one vanishing point. Simply selecting one point with the highest confidence would be random (provided that the point is on the line corresponding to the vanishing point). This situation is recognized by thresholding the parametric space by Otsu's algorithm [231] (Figure 5.5 (c)). A least-squares error line is fitted to the non-zero points in the thresholded image (Figure 5.5 (d)) and if the mean non-zero points' distance to the fitted line is below a threshold, then the one dominant motion situation is recognized. As a fallback solution, the horizon is declared to be the one horizon with zero slope and the highest confidence (Figure 5.5 (e)). A non-problematic scene with multi-direction movement is shown in Figure 5.6 for contrast; the 'line' pattern in the parametric

(a) (b) (c) (d) (e)

FIGURE 5.5: Scene with a single dominant motion direction. (a) The scene. The final horizon given by the fallback solution is marked by a red line; (b) Parametric space $\mathcal{M} \times \mathcal{Y}$ of the scene with apparent vanishing point pattern; (c) Parametric space $\mathcal{M} \times \mathcal{Y}$ after OTSU threshold computation; (d) Line fitted to the non-zero values from (c) for points' mean distance computation to 'line' pattern detection; (e) Parametric space $\mathcal{M} \times \mathcal{Y}$ with fallback solution (zero slope) marked by a red circle.

(a) (b) (c) (d) (e)

FIGURE 5.6: Scene with multi-direction movement. (e) Parametric space $\mathcal{M} \times \mathcal{Y}$ with solution marked by red circle (no fallback)

space is not present, and the fallback solution is not necessary.

5.4 Dataset — Geometric and Human Annotations

This section describes the dataset collected for testing the horizon estimation and its annotations (geometric and human).

5.4.1 Data Collection

Most of the recordings were taken from publicly available IP cameras, some recordings were captured by a camcorder. One scene was used from the PETS dataset, but it is not a very suitable one because of its short duration. One overcrowded scene was used from [232] to cover as many complex scenes as possible.

The recordings differ in many aspects – places, camera positions, daytime, scene type, duration, resolution, ... The collection includes scenes from traffic, indoors, outdoors, pedestrians, etc. Some recordings were taken during the night so different light conditions are also available. The duration of the recordings is in the range from 5 minutes to 30 hours, mean length is about 2.9 hours (details in Figure 5.7). The resolution is largely varying with the given IP camera's quality in the range from 320×240 to $1\,920 \times 1\,080$ pixels. In total, 47 different usable scenes were obtained. Some scenes were re-captured under different conditions (lighting, crowd density, ...), yielding 66 recordings in total.

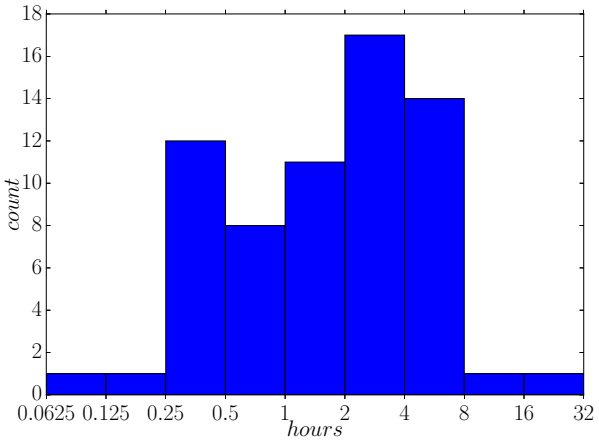

FIGURE 5.7: Histogram of video durations in the dataset.

5.4.2 Horizon Annotations

Obtaining horizon ground truth turned out to be a challenging problem mostly due to the very frequent occlusion of the natural horizon in the scenes (buildings, horizon out of frame, ...). In order to obtain a geometrical estimation of the horizon, we extracted one representative frame from the video recording and manually annotated groups of lines that are parallel in the original 3D scene (edges of a house's windows, markings on the streets, patterns in the pavings, etc.). Each of these groups of lines provides one estimated vanishing point; all vanishing points should be collinear – coincident with the line of the horizon. The horizon is obtained by using the least-squares linear regression on the set of the estimated vanishing points obtained as the minimal error intersections of the lines in the individual groups. Such obtained horizon is referenced as the 'geometric' annotation later in this text. This geometric horizon line is established for every scene of the dataset; an example of this annotation process is depicted in Figure 5.8 right.

Aside from the geometric horizon estimation, we collected horizon annotations by humans. We created a web annotation tool (Figure 5.8 left) and knowledgeable people were asked to estimate the horizon in the scene frame as precisely as possible. As described by Herdtweck and Wallraven [209], people are able to localize the horizon in a given image with small errors

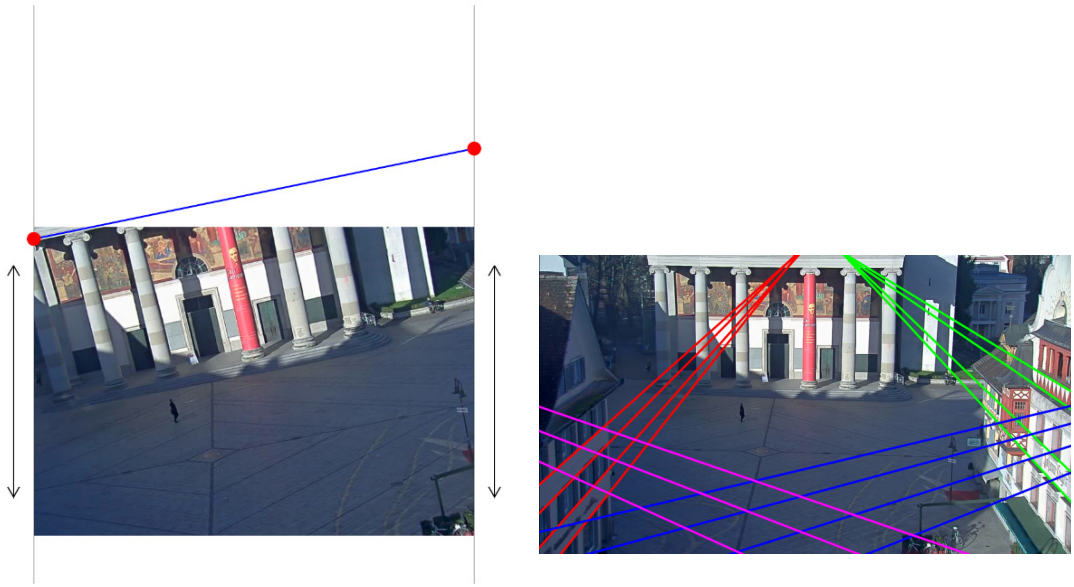

FIGURE 5.8: Scene horizon annotation principles. **left:** Web annotation tool for crowdsourced data collection. **right:** 'Geometric' ground truth annotation by using scene parallel lines.

after a short description of what the horizon really is. To prevent people from simply assuming a horizon line is always horizontal in the image (though this is common in many camera shots), the images given to the users for annotation were rotated by $\pm 20^{\circ}$ and the maximal rectangle was slightly cropped as in Figure 5.8 left.

Participants estimated the horizon by moving a visual line with markers on its sides as precisely as possible – the users could try different positions of the controlled line and look for the best match. Every participant marked 20 least annotated scenes. The annotations were filtered to rid of annotations clearly skipped or carelessly performed. Finally, 16 – 21 annotations for each of the 47 scenes are available (mean number 18.42 annotations per scene by different human subjects). Some examples of annotated scenes are depicted in Figure 5.9. It is apparent that in some scenes, it is very difficult for a human to mark the horizon (Figure 5.9 right) – mostly in cases when the horizon is 'somewhere above' the frame or totally occluded. In some scenes (Figure 5.9 left), the correlation between the horizons indicated by humans is very high.

5.4.3 Trajectories Data

Object tracking (together with the necessary video decompression) is computationally the most difficult part of the whole process of horizon estimation, and so these data are stored for faster processing and also for possible later usage of these data by other users as part of our dataset. We used our own implementation of the object tracking method proposed by Yang *et al.* [233]. For every scene in the dataset, the following information is stored:

- Scene name
- 'Geometric' annotation
- Humans' annotations

FIGURE 5.9: Examples of humans' annotations. Blue lines are individual annotations, cyan is the mean horizon location. **left:** Convenient scenes. **right:** Scenes with high variance in the annotations.

- Contours for all observations of individual trajectories

TABLE 5.1: Algorithm parameters

Name	Usage	Value
B	Section 5.3.2	45 (each bin covers 4°)
C	Section 5.3.2	10
Γ	Section 5.3.2	10.0
$\vec{\mathbf{h}}_{ref}$	Section 5.3.2	$[1,0]$ – zero slope
β	Section 5.3.3	20°
s_m	Section 5.3.3	0.5°
y_{gen}	Section 5.3.3	0.8
s_y	Section 5.3.3	1.0 px
δ_e	Section 5.3.3	5e−2
δ_c	Section 5.3.3	5e−4

5.5 Experimental Results

This section evaluates the algorithm by comparing it to the humans' and 'geometric' annotations on all 47 scenes (66 recordings) from our dataset.

5.5.1 Experimental Setup

All experiments used parameters defined in Table 5.1. Preliminary experiments showed that the computation is not very sensitive to settings of parameters δ_e and δ_c. As was mentioned in Section 5.3.1, axis-aligned bounding boxes of the observations are used for computation of observations' dimensions $\mathbf{t}(i)^{\mathbf{d}}$. Experiments with contour usage instead of axis-aligned bounding box were done but contours' dimensions incline to be more noisy.

After the preliminary experiments, the weight vector $\mathbf{w}$ (Section 5.3.3) was set to the value $\mathbf{w} = (0.7; 0.3; 0.6)$, where the observation dimensions $\mathbf{t}(i)^{\mathbf{d}}$ are sequentially: bounding box height, width, and diagonal length. The height provides the most valuable contribution to the resulting error computation (5.10) and is well usable for humans (objects that do not change dimensions rapidly during motion). Width and diagonal lengths help with computation for objects, which change their dimensions by rotation in the scene (typically vehicles).

5.5.2 Evaluation and Results

The experiments are done by comparing the proposed algorithm outputs with the humans' and the 'geometric' annotations in their values of ω (angle of horizon's slope in degrees) and ψ (vertical position of the horizon, relative to image height, i.e. 0 at image top, 1.0 at image bottom.

The left graph in Figure 5.10 displays the absolute differences between the horizons' positions ψ, comparing the algorithmic outputs to the humans' annotations (mean horizon) and to the 'geometric' references: when compared to the humans' annotations, in 80 % cases, the relative vertical position difference is under 0.112 (cca 11 % of the image height), and 95 % of cases fit under 0.144 relative vertical position difference. When compared to the 'geometric' annotations, the values of ψ are bellow 0.055 in 80 % of cases and bellow 0.083 in 95 % cases. The right plot in Figure 5.10 shows the absolute differences between the horizon angles, compared both to the humans' annotations and to the 'geometric' reference. When compared

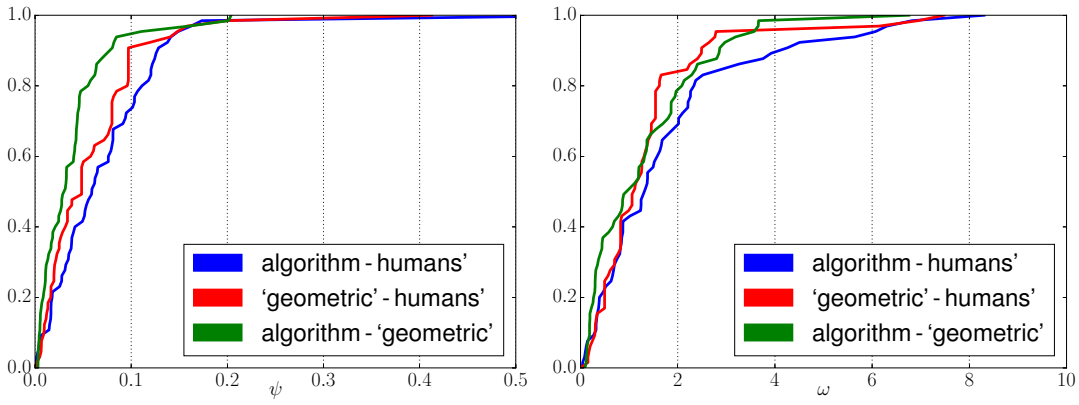

FIGURE 5.10: Cumulative histograms of differences between different methods of obtaining the horizons (human crowdsourced annotation, 'geometric' horizons, algorithmic method). **left:** differences in vertical position ψ (relative to frame height), **right:** differences in the horizon's angle ω in degrees.

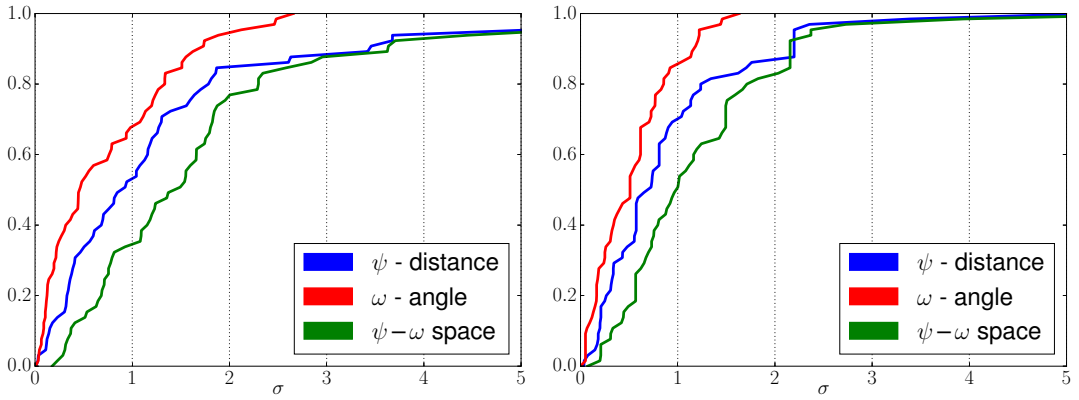

FIGURE 5.11: Cumulative histograms of output distances to the human input means in terms of σ. **left:** algorithmic horizons, **right:** 'geometric' horizons.

to the human annotations, in 80 % cases the angular error (in degrees) was below 2.33° and in 95 % cases below 5.7°. When compared to the 'geometric' annotations, in 80 % cases the value was below 2.08°, and in 95 % cases it fit under 3.3°.

The graphs in Figure 5.10 indicate that the algorithm's outputs compared to 'geometric' ground truth are on par or outperform the human 'guesses'. We, therefore, conclude that the algorithmic approach based on observing moving objects can provide the 'gist' of the scene [209, 210] used by humans for understanding an arbitrary visual scene.

For another comparison with the humans' annotations, the mean of annotation horizons is taken as the reference value and the errors are expressed in terms of standard deviation σ – see Figure 5.11 for the results. The left graph shows the difference of the algorithmically obtained horizons from the mean of human annotations; the right graph shows the distance of the 'geometric' annotations from the mean of the human annotations. It is apparent that humans can indicate the horizon quite accurately because the Mahalanobis distance in 2D $\psi-\omega$ space (as also depicted in Figure 5.12) is below 1σ in 51 % of cases and in 83 % cases bellow 2σ.

Some examples of the proposed algorithm outputs can be seen in Figure 5.12 together with the humans' and 'geometric' references.

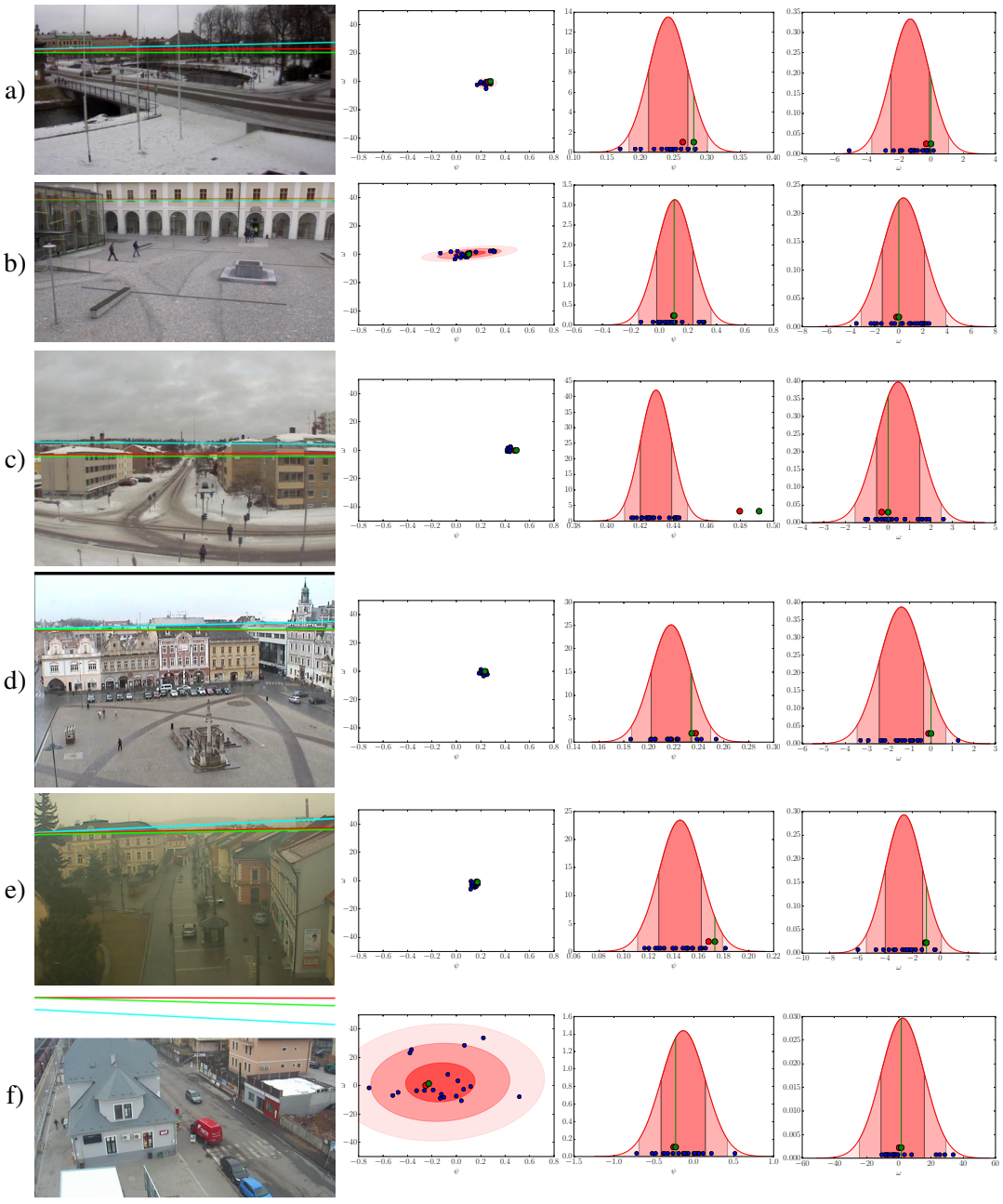

FIGURE 5.12: Selected scene samples. **camera images** include alternative horizons; *red:* 'geometric' horizon, *green:* horizon computed from the video, *cyan:* mean of human inputs. **graphs:** *blue dots* are individual human annotations, *green dot* is computed by the algorithm, *red dot* is the 'geometric' horizon, *shades of red* are parts of normal distribution of the human inputs $(1\sigma, 2\sigma, 3\sigma)$ *left:* $\psi - \omega$ plot, *center:* distribution along ψ axis, *right:* distribution along ω axis.

The main source of inaccuracies is the noise in the tracked data – occlusions of the tracked objects, pixel segmentation imprecisions, etc. Despite that, the method works with comparable accuracy as the human annotators; in difficult scenes, our method outperforms humans (with the 'geometric' horizon as the reference). One source of inaccuracy is the radial distortion of many of the cameras. In such cases, the horizons (lines) produced by our method behave similarly to the human annotations and to the geometric construction. Thus established approximate horizon can still serve the purpose of a basic understanding of the scene. The final observation is that with longer videos, the results improve (as expected) and the noise cancels out. Our method is, therefore, suitable for fixed surveillance cameras, because with more coming data, it has the chance to improve (contrary to methods only processing one frame of the video).

State-of-the-art method for horizon detection by only processing one image by Zhai *et al.* [224] fails when short line segments or curved structures are present in a scene (relies on line segments). Our proposed method does not rely on visual information thus it can also handle this type of scene. By contrast, compared to methods that only process one image, the proposed method can finish with different results for the same scene (depending on the motions happening in the scene at the given time). Figure 5.13 shows results for different recordings of the same scene.

FIGURE 5.13: Resulting detected horizons for different recordings of the same scene (different lighting conditions, daytime, present objects, ...); *red:* 'geometric' horizon, *green:* horizon computed from the video, *cyan:* mean of human inputs

5.6 Conclusions

This paper introduced an algorithm for estimating the horizon in surveillance videos based only on motion in the scene. Contrary to existing approaches, our algorithm does not assume the presence of particular objects (such as vehicles or humans) in the scene, but it works with arbitrary scenes.

We collected a set of videos from real-life web cameras, surveillance cameras, and other scenes, and make it public along with this paper. The videos in this dataset are very diverse (in terms of scale, nature of the scene, type of objects appearing, horizon position, lighting, etc.). We provide two kinds of annotations of this dataset: geometrically extracted horizons, and direct human annotations. The dataset also contains the tracks of the objects moving in the scene.

Our algorithm manages to get the 'gist' of the scene, which could help other tasks of computer vision (as it has been shown that it helps humans in their understanding). The experiments show that the accuracy achieved by our solution is comparable to the performance of human annotators; some scenes even confused the human annotators so much that our algorithm outperformed humans. Our purpose was to show that this task is possible to solve and to establish a baseline for further development.

Chapter 6

Proposed Future Work

The results presented in Section 3.3.2 are a considerable improvement over the previous state-of-the-art method [32], but they are still not totally satisfying: both the accuracy of the ground truth (around 1 % error) and the results of the calibration themselves (around 4 % error evaluated by the mentioned ground truth). We have carried out preliminary experiments which should lead to considerable improvement of the accuracy. The calibration methodology presented in the article remains the same; the improvements are centered in providing better inputs to the algorithms: providing both better ground truth measurements and even more importantly, better landmarks on the vehicle.

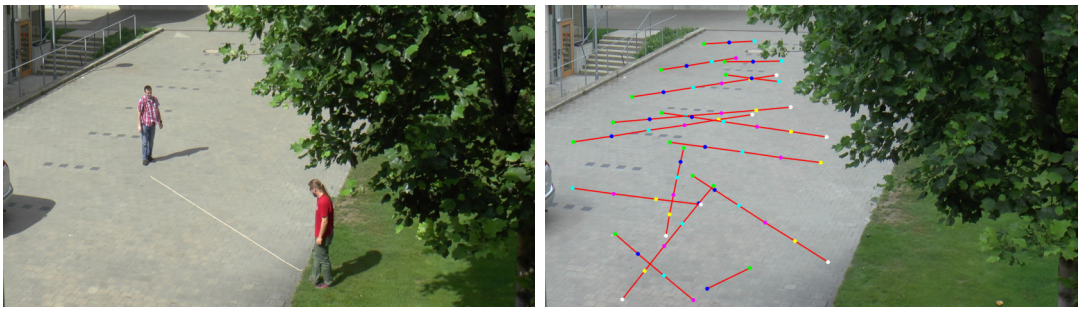

FIGURE 6.1: Experiments with more precise ground truth calibration measurements. *left:* Two people place a rope with regular marks (five 1 m sections in this case) into multiple locations in front of the camera. *right:* In a short and uncomplicated process, a high number of precise measurements can be obtained — for each rope placement, one frame of the video gives several very accurate distance measurements in the ground plane by manual or automatic recognition of the distance marks.

Firstly, Figure 6.1 illustrates the new approach to ground truth measurements. The motivation for the new design is twofold. The measurements done with a precisely constructed straight rope are more precise because they do not rely on distances between ambiguous natural elements in the scene. Besides, this approach allows us to obtain a multitude of measurements with small effort and in a short time. Each rope placement provides several (five in the shown case) measurements along one straight line, and the rope can be easily placed into multiple locations and orientations. The experiment captured in Figure 6.1 resulted in the ground-truth calibration of around 0.45 % (cross-validated on the rope measurements).

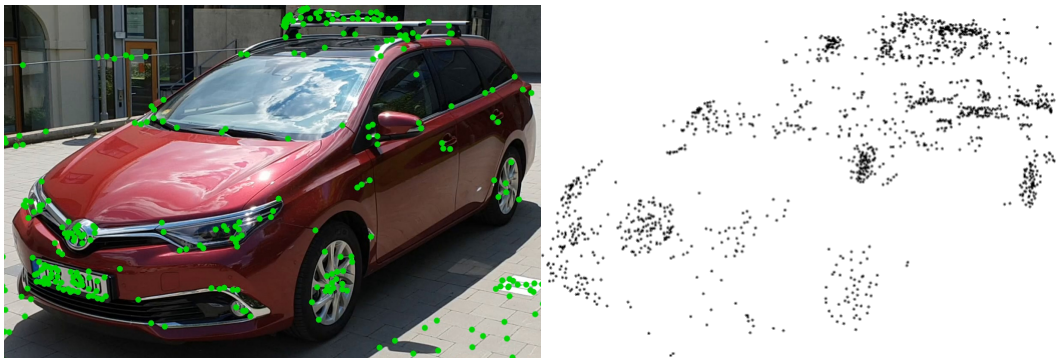

FIGURE 6.2: SfM reconstruction of a particular vehicle model, later used for accurate landmark localization. *left:* one frame of the source video, with the keypoints marked by green, *right:* reconstructed point cloud; points are candidate landmarks for the calibration.

FIGURE 6.3: Observations of a vehicle of known type, whose landmarks can be obtained accurately and their mutual distances are reconstructed precisely (Figure 6.2).

Secondly, the keypoints/landmarks on the vehicles can be extracted specifically for the given type of the observed vehicle, not by the generic (and fairly imprecise) extractor used now [118]. The true 3D distances $\hat{\delta}(\mathbf{c}_i, a, b)$ are already make & model specific (Section 3.3.1), and therefore assuming fine-grained recognition of the vehicle ([113, 114]) does not constitute a new requirement. We made a detailed reconstruction of one vehicle (Toyota Auris SW 2017) from a walk-around video by using an existing Structure-from-Motion solution *OpenMVG* [234], see Figure 6.2. This vehicle moved randomly in the scene and 25 observations varying in the vehicle's location and orientation were selected, see Figure 6.3. The landmarks/keypoints were manually extracted and their real-world 3D distances $\hat{\delta}(\mathbf{c}_i, a, b)$ were obtained from the point cloud reconstructed by the SfM. The calibration obtained by our algorithm from this input achieved an error of 0.79 % (evaluated by the new ground truth, Figure 6.1). We are working on making this process fully automatic, but this preliminary experiment shows that the algorithm presented in this article can promise very usable accuracy when the input data is sufficiently precise. The purpose of this section is to show that such precise input data indeed could be obtained.

Chapter 7

Conclusion

This thesis presents my work during my Ph.D. study. As the main core, I consider three proposed methods for automatic camera calibration based on detecting landmarks on rigid objects (in my case, vehicles). These methods are described in Chapter 3, which is reduced compared to the original papers, as there are many common things to all methods. The goal is better readability and a more straightforward understanding of the methods.

Individual methods were compared to different datasets in the respective papers. Later after their publication, our *BrnoCarPark* dataset was published, that since then, became a good benchmark dataset for similar methods. The dataset is one of the main contributions of my Ph.D. study as it is publicly available and it serves other researchers with their work. I also provide the implementation of all the methods publicly available, and anyone can use the code as they want.

As the dataset is public, I was able to evaluate all my methods on the new *BrnoCarPark* dataset together with *BrnoCarSpeed* (published by my colleagues). It seems that the *LandmarksCalib* method is the most stable and it can handle different types of scenes. I also evaluated different possible combinations of the methods, where the main idea is using a fast method for a coarse estimation and further more precise computation by a slower method; this idea did not seem to bring additional gains.

At the beginning of my study, I tried to follow up on my master's thesis and made a method for horizon estimation. Still, the direction of landmarks calibration seemed to me much more interesting. During the development of the calibration methods, I started to co-work with my colleagues in the ITS area a co-authored some of the works. One of the results of this cooperation was participation in workshops — mainly the *AI City Challenge*.

Work on various projects was also an integral part of my Ph.D. study. Although the work could not be published in the form of research papers (often commercial/restricted usage by companies), it gave me a lot of knowledge and experience and the work led to production use of my research and development.

My Publications

[1] Vojtěch Bartl and Adam Herout. “OptInOpt: Dual Optimization for Automatic Camera Calibration by Multi-Target Observations”. In: *International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. 2019.

[2] Vojtěch Bartl, Roman Juránek, Jakub Špaňhel, and Adam Herout. “PlaneCalib: Automatic Camera Calibration by Multiple Observations of Rigid Objects on Plane”. In: *Digital Image Computing: Techniques and Applications (DICTA)*. 2020.

[3] Vojtech Bartl, Jakub Špaňhel, Petr Dobeš, Roman Juránek, and Adam Herout. “Automatic Camera Calibration by Landmarks on Rigid Objects”. In: *Machine Vision and Applications (MVAP)*. 2021.

[4] Vojtěch Bartl and Adam Herout. “Fully Automatic Horizon Estimation for Surveillance Cameras”. In: *Digital Image Computing: Techniques and Applications (DICTA)*. 2017.

[5] Jakub Špaňhel, Jakub Sochor, Roman Juranek, Petr Dobeš, Vojtěch Bartl, and Adam Herout. “Learning Feature Aggregation in Temporal Domain for Re-Identification”. In: *Computer Vision and Image Understanding (CVIU)*. 2020.

[6] Petr Dobeš, Jakub Špaňhel, Vojtěch Bartl, Roman Juránek, and Adam Herout. “Density-Based Vehicle Counting with Unsupervised Scale Selection”. In: *Digital Image Computing: Techniques and Applications (DICTA)*. 2020.

[7] Jakub Špaňhel, Vojtěch Bartl, Roman Juranek, and Adam Herout. “Vehicle Re-Identifiation and Multi-Camera Tracking in Challenging City-Scale Environment”. In: *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2019.

[8] Ján Folenta, Jakub Špaňhel, Vojtěch Bartl, and Adam Herout. “Determining Vehicle Turn Counts at Multiple Intersections by Separated Vehicle Classes Using CNNs”. In: *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2020.

[9] Vojtěch Bartl, Jakub Špaňhel, and Adam Herout. “PersonGONE: Image Inpainting for Automated Checkout Solution”. In: *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2022.

[10] Benjamin Kiefer, Matej Kristan, Janez Perš, Lojze Žust, Fabio Poiesi, Fabio Augusto de Alcantara Andrade, Alexandre Bernardino, Matthew Dawkins, Jenni Raitoharju, Yitong Quan, Adem Atmaca, Timon Höfer, Qiming Zhang, Yufei Xu, Jing Zhang, Dacheng Tao, Lars Sommer, Raphael Spraul, Hangyue Zhao, Hongpu Zhang, Yanyun Zhao, Jan Lukas Augustin, Eui-ik Jeon, Impyeong Lee, Luca Zedda, Andrea Loddo, Cecilia Di Ruberto, Sagar Verma, Siddharth Gupta, Shishir Muralidhara,

Niharika Hegde, Daitao Xing, Nikolaos Evangeliou, Anthony Tzes, Vojtěch Bartl, Jakub Špaňhel, Adam Herout, Neelanjan Bhowmik, Toby P. Breckon, Shivanand Kundargi, Tejas Anvekar, Chaitra Desai, Ramesh Ashok Tabib, Uma Mudengudi, Arpita Vats, Yang Song, Delong Liu, Yonglin Li, Shuman Li, Chenhao Tan, Long Lan, Vladimir Somers, Christophe De Vleeschouwer, Alexandre Alahi, Hsiang-Wei Huang, Cheng-Yen Yang, Jenq-Neng Hwang, Pyong-Kun Kim, Kwangju Kim, Kyoungoh Lee, Shuai Jiang, Haiwen Li, Zheng Ziqiang, Tuan-Anh Vu, Hai Nguyen-Truong, Sai-Kit Yeung, Zhuang Jia, Sophia Yang, Chih-Chung Hsu, Xiu-Yu Hou, Yu-An Jhang, Simon Yang, and Mau-Tsuen Yang. "1st Workshop on Maritime Computer Vision (MaCVi) 2023: Challenge Results". In: *Winter Conference on Applications of Computer Vision (WACV)*. 2023.

Bibliography

[11] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. *End-to-End Object Detection with Transformers*. 2020.

[12] Ting Chen, Saurabh Saxena, Lala Li, David J Fleet, and Geoffrey Hinton. *Pix2seq: A Language Modeling Framework for Object Detection*. 2021.

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems (NIPS)*. 2015.

[14] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767* (2018).

[15] Joseph Redmon and Ali Farhadi. "YOLO9000: better, faster, stronger". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[16] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. "Simple Online and Realtime Tracking". In: *International Conference on Image Processing (ICIP)*. 2016.

[17] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple Online and Realtime Tracking with a Deep Association Metric". In: *International Conference on Image Processing (ICIP)*. 2017.

[18] Glenn Jocher. *YOLOv5*. online. 2020. URL: https://github.com/ultralytics/yolov5.

[19] Mengde Xu, Zheng Zhang, Han Hu, Jianfeng Wang, Lijuan Wang, Fangyun Wei, Xiang Bai, and Zicheng Liu. "End-to-End Semi-Supervised Object Detection with Soft Teacher". In: *International Conference on Computer Vision (ICCV)* (2021).

[20] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. "SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers". In: *Neural Information Processing Systems (NeurIPS)*. 2021.

[21] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. "Pyramid Scene Parsing Network". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[22] Jiankang Deng, Jia Guo, Yuxiang Zhou, Jinke Yu, Irene Kotsia, and Stefanos Zafeiriou. *RetinaFace: Single-stage Dense Face Localisation in the Wild*. 2019.

[23] Axel Sauer, Elie Aljalbout, and Sami Haddadin. "Tracking Holistic Object Representations". In: *British Machine Vision Conference (BMVC)*. 2019.

[24] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip HS Torr. "Fast Online Object Tracking and Segmentation: A Unifying Approach". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[25] Rui Liu, Hanming Deng, Yangyi Huang, Xiaoyu Shi, Lewei Lu, Wenxiu Sun, Xiaogang Wang, Jifeng Dai, and Hongsheng Li. "FuseFormer: Fusing Fine-Grained Information in Transformers for Video Inpainting". In: *International Conference on Computer Vision (ICCV)*. 2021.

[26] Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. "Resolution-robust Large Mask Inpainting with Fourier Convolutions". In: *arXiv preprint arXiv:2109.07161* (2021).

[27] Zhen Li, Cheng-Ze Lu, Jianhua Qin, Chun-Le Guo, and Ming-Ming Cheng. "Towards An End-to-End Framework for Flow-Guided Video Inpainting". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[28] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. "SwinIR: Image Restoration Using Swin Transformer". In: *arXiv preprint arXiv:2108.10257* (2021).

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2023.

[30] Fuwen Tan, Jiangbo Yuan, and Vicente Ordonez. "Instance-level Image Retrieval using Reranking Transformers". In: *International Conference on Computer Vision (ICCV)*. 2021.

[31] Jianyuan Guo, Kai Han, Han Wu, Chang Xu, Yehui Tang, Chunjing Xu, and Yunhe Wang. *CMT: Convolutional Neural Networks Meet Vision Transformers*. 2021.

[32] Romil Bhardwaj, Gopi Krishna Tummala, Ganesan Ramalingam, Ramachandran Ramjee, and Prasun Sinha. "AutoCalib: Automatic Traffic Camera Calibration at Scale". In: *International Conference on Systems for Energy-Efficient Built Environments (BuildSys 2017)*. 2017.

[33] Talha Hanif Butt and Murtaza Taj. "Camera Calibration Through Camera Projection Loss". In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2022.

[34] Talha Hanif Butt and Murtaza Taj. "Multi-task Learning for Camera Calibration". In: *arXiv preprint arXiv:2211.12432* (2022).

[35] Gabriel Van Zandycke, Vladimir Somers, Maxime Istasse, Carlo Del Don, and Davide Zambrano. "DeepSportradar-v1: Computer Vision Dataset for Sports Understanding with High Quality Annotations". In: *Workshop on Multimedia Content Analysis in Sports*. ACM, 2022.

[36] Zhengyou Zhang. "A Flexible New Technique for Camera Calibration". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2000).

[37] Xiaoqiao Meng and Zhanyi Hu. "A New Easy Camera Calibration Technique Based on Circular Points". In: *Pattern Recognition* (2003).

[38] Zhengyou Zhang. "Camera Calibration with One-dimensional Objects". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2004).

[39] Pavel Rojtberg and Arjan Kuijper. “Efficient Pose Selection for Interactive Camera Calibration”. In: *International Symposium on Mixed and Augmented Reality (ISMAR)*. 2018.

[40] Songyou Peng and Peter Sturm. “Calibration Wizard: A Guidance System for Camera Calibration Based on Modelling Geometric and Corner Uncertainty”. In: *International Conference on Computer Vision (ICCV)*. 2019.

[41] Imran N. Junejo and Hassan Foroosh. “Robust Auto-Calibration from Pedestrians”. In: *International Conference on Video and Signal Based Surveillance*. 2006.

[42] Worapan Kusakunniran, Hongdong Li, and Jian Zhang. “A Direct Method to Self-Calibrate a Surveillance Camera by Observing a Walking Pedestrian”. In: *Digital Image Computing: Techniques and Applications (DICTA)*. 2009.

[43] Fengjun Lv, Tao Zhao, and R. Nevatia. “Self-Calibration of a Camera From Video of a Walking Human”. In: *International Conference on Pattern Recognition (ICPR)*. 2002.

[44] Fengjun Lv, Tao Zhao, and R. Nevatia. “Camera Calibration From Video of a Walking Human”. In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2006).

[45] Guido M. Y. E. Brouwers, Matthijs H. Zwemer, Rob G. J. Wijnhoven, and Peter H. N. de With. “Automatic Calibration of Stationary Surveillance Cameras in the Wild”. In: *European Conference on Computer Vision Workshops (ECCVW)*. 2016.

[46] Cristina Maduro, Katherine Batista, Paulo Peixoto, and Jorge Batista. “Estimation of Vehicle Velocity and Traffic Intensity Using Rectified Images”. In: *International Conference on Image Processing (ICIP)*. 2008.

[47] Adi Nurhadiyatna, Benny Hardjono, Ari Wibisono, Immaculate Sina, Wisnu Jatmiko, M.Anwar Ma’sum, and Petrus Mursanto. “Improved Vehicle Speed Estimation Using Gaussian Mixture Model and Hole Filling Algorithm”. In: *Advanced Computer Science and Information Systems (ICACSIS), 2013 International Conference on*. 2013.

[48] Diogo C. Luvizon, Bogdan Tomoyoki Nassu, and Rodrigo Minetto. “Vehicle Speed Estimation by License Plate Detection and Tracking”. In: *Acoustics, Speech and Signal Processing (ICASSP)*. 2014.

[49] F.W. Cathey and Daniel Dailey. “A Novel Technique to Dynamically Measure Vehicle Speed Using Uncalibrated Roadway Cameras”. In: *Intelligent Vehicles Symposium*. 2005.

[50] Lazaros Grammatikopoulos, George Karras, and Elli Petsa. “Automatic Estimation of Vehicle Speed from Uncalibrated Video Sequences”. In: *Proceedings of International Symposium on Modern Technologies, Educationand Profeesional Practice in Geodesy and Related Fields*. 2005.

[51] Xiao Chen He and N. H C Yung. “A Novel Algorithm for Estimating Vehicle Speed from Two Consecutive Images”. In: *Workshop on Applications of Computer Vision (WACV)*. 2007.

[52] Viet-Hoa Do, Le-Hoa Nghiem, Ngoc Pham Thi, and Nam Pham Ngoc. "A Simple Camera Calibration Method for Vehicle Velocity Estimation". In: *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. 2015.

[53] Xinhua You and Yuan Zheng. "An Accurate and Practical Calibration Method for Roadside Camera Using Two Vanishing Points". In: *Neurocomputing* (2016).

[54] Daniel Dailey, Fritz W. Cathey, and Suree Pumrin. "An Algorithm to Estimate Mean Traffic Speed Using Uncalibrated Cameras". In: *Transactions on Intelligent Transportation Systems (T-ITS)* (2000).

[55] Todd N. Schoepflin and Daniel J. Dailey. "Dynamic Camera Calibration of Roadside Traffic Management Cameras for Vehicle Speed Estimation". In: *Transactions on Intelligent Transportation Systems (T-ITS)* (2003).

[56] Markéta Dubská, Jakub Sochor, and Adam Herout. "Automatic Camera Calibration for Traffic Understanding". In: *British Machine Vision Conference (BMVC)*. 2014.

[57] Jakub Sochor, Roman Juránek, and Adam Herout. "Traffic Surveillance Camera Calibration by 3D Model Bounding Box Alignment for Accurate Vehicle Speed Measurement". In: *Computer Vision and Image Understanding (CVIU)* (2017).

[58] Jinhui Lan, Jian Li, Guangda Hu, Bin Ran, and Ling Wang. "Vehicle Speed Measurement Based on Gray Constraint Optical Flow Algorithm". In: *Optik – International Journal for Light and Electron Optics* (2014).

[59] David Fernández Llorca, Camila Salinas, M. Jimenez, I. Parra, A. G. Morcillo, R. Izquierdo, Javier Lorenzo, and Miguel Ángel Sotelo. "Two-camera Based Accurate Vehicle Speed Measurement Using Average Speed at a Fixed Point". In: *Conference on Intelligent Transportation Systems (ITSC)*. 2016.

[60] Patryk Filipiak, Bartlomiej Golenko, and Cezary Dolega. "NSGA-II Based Auto-Calibration of Automatic Number Plate Recognition Camera for Vehicle Speed Measurement". In: *EvoApplications 2016*. Springer International Publishing, 2016.

[61] Markéta Dubská and Adam Herout. "Real Projective Plane Mapping for Detection of Orthogonal Vanishing Points". In: *British Machine Vision Conference (BMVC)*. 2013.

[62] Viktor Kocur and Milan Ftáčnik. "Traffic Camera Calibration via Vehicle Vanishing Point Detection". In: *Artificial Neural Networks and Machine Learning – ICANN 2021*. 2021.

[63] Longyin Wen, Dawei Du, Zhaowei Cai, Zhen Lei, Ming-Ching Chang, Honggang Qi, Jongwoo Lim, Ming-Hsuan Yang, and Siwei Lyu. "UA-DETRAC: A New Benchmark and Protocol for Multi-Object Detection and Tracking". In: *Computer Vision and Image Understanding (CVIU)* (2020).

[64] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. *Microsoft COCO: Common Objects in Context*. 2014.

[65] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *International Journal of Computer Vision (IJCV)* 111 (2015).

[66] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[67] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. *BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning*. 2020.

[68] Xibin Song, Peng Wang, Dingfu Zhou, Rui Zhu, Chenye Guan, Yuchao Dai, Hao Su, Hongdong Li, and Ruigang Yang. "ApolloCar3D: A Large 3D Car Instance Understanding Benchmark for Autonomous Driving". In: *Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).

[69] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulò, and Peter Kontschieder. "The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes". In: *International Conference on Computer Vision (ICCV)*. 2017.

[70] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. "nuScenes: A Multimodal Dataset for Autonomous Driving". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[71] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv preprint arXiv:1409.1556* abs/1409.1556 (2014).

[72] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[73] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the Inception Architecture for Computer Vision". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[74] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[75] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[76] Mingxing Tan and Quoc V. Le. *EfficientNetV2: Smaller Models and Faster Training*. 2021.

[77] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. "Aggregated Residual Transformations for Deep Neural Networks". In: *arXiv preprint arXiv:1611.05431* (2016).

[78] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size". In: *arXiv:1602.07360* (2016).

[79] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. “YOLACT: Real-time Instance Segmentation”. In: *International Conference on Computer Vision (ICCV)*. 2019.

[80] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. “YOLACT++: Better Real-time Instance Segmentation”. In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2020).

[81] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016.

[82] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. “Yolov4: Optimal Speed and Accuracy of Object Detection”. In: *arXiv preprint arXiv:2004.10934* (2020).

[83] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, and Xiaolin Wei. *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*. 2022.

[84] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022.

[85] Xiang Long, Kaipeng Deng, Guanzhong Wang, Yang Zhang, Qingqing Dang, Yuan Gao, Hui Shen, Jianguo Ren, Shumin Han, Errui Ding, and Shilei Wen. *PP-YOLO: An Effective and Efficient Implementation of Object Detector*. 2020.

[86] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. *You Only Learn One Representation: Unified Network for Multiple Tasks*. 2021.

[87] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. “YOLOX: Exceeding YOLO Series in 2021”. In: *arXiv preprint arXiv:2107.08430* (2021).

[88] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. “SSD: Single Shot Multibox Detector”. In: *European Conference on Computer Vision (ECCV)*. 2016.

[89] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. *Focal Loss for Dense Object Detection*. 2018.

[90] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. “CenterNet: Keypoint Triplets for Object Detection”. In: *International Conference on Computer Vision (ICCV)*. 2019.

[91] Mingxing Tan, Ruoming Pang, and Quoc V. Le. *EfficientDet: Scalable and Efficient Object Detection*. 2020.

[92] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[93] Ross Girshick. “Fast R-CNN”. In: *The IEEE International Conference on Computer Vision (ICCV)*. 2015.

[94] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. “Mask R-CNN”. In: *International Conference on Computer Vision (ICCV)*. 2017.

[95] Alexander Kirillov, Ross B. Girshick, Kaiming He, and Piotr Dollár. "Panoptic Feature Pyramid Networks". In: *arXiv preprint arXiv:1901.02446* (2019).

[96] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. *Feature Pyramid Networks for Object Detection*. 2017.

[97] Siyuan Qiao, Liang-Chieh Chen, and Alan Yuille. "DetectoRS: Detecting Objects with Recursive Feature Pyramid and Switchable Atrous Convolution". In: *arXiv preprint arXiv:2006.02334* (2020).

[98] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations (ICLR)*. 2021.

[99] Hwanjun Song, Deqing Sun, Sanghyuk Chun, Varun Jampani, Dongyoon Han, Byeongho Heo, Wonjae Kim, and Ming-Hsuan Yang. "ViDT: An Efficient and Effective Fully Transformer-based Object Detector". In: *International Conference on Learning Representation (ICLR)*. 2022.

[100] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M. Ni, and Heung-Yeung Shum. *DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection*. 2022.

[101] Jakub Sochor, Jakub Špaňhel, and Adam Herout. "BoxCars: Improving Fine-Grained Recognition of Vehicles Using 3-D Bounding Boxes in Traffic Surveillance". In: *IEEE Transactions on Intelligent Transportation Systems (T-ITS)* (2018).

[102] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. "A Large-scale Car Dataset for Fine-grained Categorization and Verification". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[103] Prafful Kumar Khoba, Chirag Parikh, Rohit Saluja, Ravi Kiran Sarvadevabhatla, and C.V. Jawahar. "A Fine-Grained Vehicle Detection (FGVD) Dataset for Unconstrained Roads". In.

[104] Hongye Liu, Yonghong Tian, Yaowei Wang, Lu Pang, and Tiejun Huang. "Deep Relative Distance Learning: Tell the Difference Between Similar Vehicles". In: *Conference on Computer Vision and Pattern Recognition (CVPR*. 2016.

[105] Marcel Simon and Erik Rodner. "Neural Activation Constellations: Unsupervised Part Model Discovery with Convolutional Networks". In: *International Conference on Computer Vision (ICCV)*. 2015.

[106] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. "Bilinear CNN Models for Fine-grained Visual Recognition". In: *International Conference on Computer Vision (ICCV)*. 2015.

[107] Yang Gao, Oscar Beijbom, Ning Zhang, and Trevor Darrell. "Compact Bilinear Pooling". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[108] Di Lin, Xiaoyong Shen, Cewu Lu, and Jiaya Jia. "Deep LAC: Deep Localization, Alignment and Classification for Fine-Grained Recognition". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[109] Greg Pearce and Nick Pears. “Automatic Make and Model Recognition from Frontal Images of Cars”. In: *Conference on Advanced Video and Signal Based Surveillance (AVSS)*. 2011.

[110] Bailing Zhang. “Classification and Identification of Vehicle Type and Make by Cortex-like Image Descriptor HMAX”. In: *International Journal of Computational Vision and Robotics (IJCVR)* (2014).

[111] Yen-Liang Lin, Vlad I. Morariu, Winston Hsu, and Larry S. Davis. “Jointly Optimizing 3D Model Fitting and Fine-Grained Classification”. In: *European Conference on Computer Vision (ECCV)*. 2014.

[112] Hongye Liu, Yonghong Tian, Yaowei Yang, Lu Pang, and Tiejun Huang. “Deep Relative Distance Learning: Tell the Difference Between Similar Vehicles”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[113] Jakub Sochor, Adam Herout, and Jiri Havel. “BoxCars: 3D Boxes as CNN Input for Improved Fine-Grained Vehicle Recognition”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[114] Jakub Sochor, Jakub Špaňhel, and Adam Herout. “BoxCars: Improving Fine-Grained Recognition of Vehicles using 3D Bounding Boxes in Traffic Surveillance”. In: *arXiv preprint arXiv:1703.00686* (2017).

[115] Hamed Pirsiavash, Deva Ramanan, and Charless C. Fowlkes. “Bilinear Classifiers for Visual Recognition”. In: *Advances in Neural Information Processing Systems (NIPS)*. Curran Associates, Inc., 2009.

[116] Anqi Hu, Zhengxing Sun, Qian Li, Yechao Xu, Yihuan Zhu, and Sheng Zhang. “Fine-grained Traffic Video Vehicle Recognition Based Orientation Estimation and Temporal Information”. In: *Multimedia Tools and Applications* (2023).

[117] Ju He, Jie-Neng Chen, Shuai Liu, Adam Kortylewski, Cheng Yang, Yutong Bai, and Changhu Wang. “TransFG: A Transformer Architecture for Fine-grained Recognition”. In: *Conference on Artificial Intelligence*. 2022.

[118] Zhongdao Wang, Luming Tang, Xihui Liu, Zhuliang Yao, Shuai Yi, Jing Shao, Junjie Yan, Shengjin Wang, Hongsheng Li, and Xiaogang Wang. “Orientation Invariant Feature Embedding and Spatial Temporal Regularization for Vehicle Re-Identification”. In: *International Conference on Computer Vision (ICCV)*. 2017.

[119] Xinchen Liu, Wu Liu, Huadong Ma, and Huiyuan Fu. “Large-scale Vehicle Re-Identification in Urban Surveillance Videos”. In: *International Conference on Multimedia and Expo (ICME)*. 2016.

[120] Sheng Jin, Lumin Xu, Jin Xu, Can Wang, Wentao Liu, Chen Qian, Wanli Ouyang, and Ping Luo. “Whole-Body Human Pose Estimation in the Wild”. In: *European Conference on Computer Vision (ECCV)*. 2020.

[121] Alejandro Newell, Kaiyu Yang, and Jia Deng. “Stacked Hourglass Networks for Human Pose Estimation”. In: *European Conference on Computer Vision (ECCV)*. 2016.

[122] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. “Deep High-Resolution Representation Learning for Human Pose Estimation”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[123] Zhendong Yang, Ailing Zeng, Chun Yuan, and Yu Li. "Effective Whole-body Pose Estimation with Two-stages Distillation". In: *arXiv preprint arXiv:2307.15880* (2023).

[124] Yufei Xu, Jing Zhang, Qiming Zhang, and Dacheng Tao. "ViTPose: Simple Vision Transformer Baselines for Human Pose Estimation". In: *Advances in Neural Information Processing Systems*. 2022.

[125] Kai-Tai Song and Jen-Chao Tai. "Dynamic Calibration of Pan–Tilt–Zoom Cameras for Traffic Monitoring". In: *Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* (2006).

[126] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications To Image Analysis and Automated Cartography". In: (1981).

[127] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. "EPnP: An Accurate O(n) Solution to the PnP Problem". In: *International Journal of Computer Vision (IJCV)* (2008).

[128] Joel A. Hesch and Stergios I. Roumeliotis. "A Direct Least-Squares (DLS) method for PnP". In: *International Conference on Computer Vision (ICCV)*. 2011.

[129] Yinqiang Zheng, Yubin Kuang, Shigeki Sugimoto, Kalle Åström, and Masatoshi Okutomi. "Revisiting the PnP Problem: A Fast, General and Optimal Solution". In: *International Conference on Computer Vision (ICCV)*. 2013.

[130] Laurent Kneip, Hongdong Li, and Yongduek Seo. "UPnP: An Optimal O(n) Solution to the Absolute Pose Problem with Universal Applicability". In: *European Conference on Computer Vision (ECCV)*. 2014.

[131] Adrian Penate-Sanchez, Juan Andrade-Cetto, and Francesc Moreno-Noguer. "Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2013).

[132] Yinqiang Zheng, Shigeki Sugimoto, Imari Sato, and Masatoshi Okutomi. "A General and Simple Method for Camera Pose and Focal Length Determination". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[133] Yinqiang Zheng and Laurent Kneip. "A Direct Least-Squares Solution to the PnP Problem with Unknown Focal Length". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[134] Roman Juránek, Adam Herout, Markéta Dubská, and Pavel Zemčík. "Real-Time Pose Estimation Piggybacked on Object Detection". In: *International Conference on Computer Vision (ICCV)*. 2015.

[135] Jakub Sochor, Roman Juránek, Jakub Špaňhel, Lukáš Maršík, Adam Široký, Adam Herout, and Pavel Zemčík. "Comprehensive Data Set for Automatic Single Camera Visual Speed Measurement". In: *Transactions on Intelligent Transportation Systems (T-ITS)* (2018).

[136] Rainer Storn and Kenneth Price. "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces". In: *Journal of Global Optimization* (1997).

[137] John C. Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *Journal of Machine Learning Research* (2011).

[138] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (2014).

[139] Dong C. Liu and Jorge Nocedal. “On the Limited Memory BFGS Method for Large Scale Optimization”. In: *Mathematical Programming* (1989).

[140] Jason P. de Villiers, F. Wilhelm Leuschner, and Ronelle Geldenhuys. “Centi-pixel Accurate Real-time Inverse Distortion Correction”. In: *Optomechatronic Technologies 2008*. 2008.

[141] Faisal Bukhari and Matthew Dailey. “Automatic Radial Distortion Estimation from a Single Image”. In: *Journal of Mathematical Imaging and Vision* (2013).

[142] Jakub Sochor, Jakub Špaňhel, Roman Juránek, Petr Dobeš, and Adam Herout. “Graph@FIT Submission to the NVIDIA AI City Challenge 2018”. In: *Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2018.

[143] Christian Szegedy, Scott Reed, Dumitru Erhan, Dragomir Anguelov, and Sergey Ioffe. “Scalable, High-Quality Object Detection”. In: *arXiv preprint arXiv:1412.1441* (2014).

[144] Yi Wei, Nenghui Song, Lipeng Ke, Ming-Ching Chang, and Siwei Lyu. “Street object detection / tracking for AI city traffic analysis”. In: *SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)* (2017).

[145] Shan Du, Mahmoud Ibrahim, Mohamed Shehata, and Wael Badawy. “Automatic license plate recognition (ALPR): A state-of-the-art review”. In: *Circuits and Systems for Video Technology* (2013).

[146] Konrad Kluwak, Jakub Segen, Marek Kulbacki, Aldona Drabik, and Konrad Wojciechowski. “ALPR - Extension to Traditional Plate Recognition Methods”. In: *Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14–16, 2016, Proceedings, Part II*. 2016.

[147] Ying Wen, Yue Lu, Jingqi Yan, Zhenyu Zhou, Karen M von Deneen, and Pengfei Shi. “An Algorithm for License Plate Recognition Applied to Intelligent Transportation System”. In: *Transactions on Intelligent Transportation Systems (T-ITS)* (2011).

[148] Clemens Arth, Christian Leistner, and Horst Bischof. “Object Reacquisition and Tracking in Large-scale Smart Camera Networks”. In: *International Conference on Distributed Smart Cameras*. 2007.

[149] Rogerio Schmidt Feris, Behjat Siddiquie, James Petterson, Yun Zhai, Ankur Datta, Lisa M Brown, and Sharath Pankanti. “Large-scale Vehicle Detection, Indexing, and Search in Urban Surveillance Videos”. In: *Transactions on Multimedia* (2012).

[150] Dominik Zapletal and Adam Herout. “Vehicle Re-Identification for Automatic Video Traffic Surveillance”. In: *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW*. 2016.

[151] Hongye Liu, Yonghong Tian, Yaowei Yang, Lu Pang, and Tiejun Huang. "Deep Relative Distance Learning: Tell the Difference Between Similar Vehicles". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[152] Yantao Shen, Tong Xiao, Hongsheng Li, Shuai Yi, and Xiaogang Wang. "Learning Deep Neural Networks for Vehicle Re-ID With Visual-Spatio-Temporal Path Proposals". In: *International Conference on Computer Vision (ICCV)*. 2017.

[153] Zhongdao Wang, Luming Tang, Xihui Liu, Zhuliang Yao, Shuai Yi, Jing Shao, Junjie Yan, Shengjin Wang, Hongsheng Li, and Xiaogang Wang. "Orientation Invariant Feature Embedding and Spatial Temporal Regularization for Vehicle Re-Identification". In: *International Conference on Computer Vision (ICCV)*. 2017.

[154] Ke Yan, Yonghong Tian, Yaowei Wang, Wei Zeng, and Tiejun Huang. "Exploiting Multi-Grain Ranking Constraints for Precisely Searching Visually-Similar Vehicles". In: *International Conference on Computer Vision (ICCV)*. 2017.

[155] Yiheng Zhang, Dong Liu, and Zheng-Jun Zha. "Improving Triplet-wise Training of Convolutional Neural Network for Vehicle Re-Identification". In: *International Conference on Multimedia and Expo (ICME)*. 2017.

[156] Xinchen Liu, Wu Liu, Tao Mei, and Huadong Ma. "A Deep Learning-Based Approach to Progressive Vehicle Re-identification for Urban Surveillance". In: *European Conference on Computer Vision (ECCV)*. 2016.

[157] Ratnesh Kumar, Edwin Weill, Farzin Aghdasi, and Parthsarathy Sriram. "Vehicle Re-Identification: An Efficient Baseline Using Triplet Embedding". In: *arXiv preprint arXiv:1901.01015* (2019).

[158] De Cheng, Yihong Gong, Sanping Zhou, Jinjun Wang, and Nanning Zheng. "Person Re-Identification by Multi-Channel Parts-Based CNN With Improved Triplet Loss Function". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[159] Alexander Hermans, Lucas Beyer, and Bastian Leibe. *In Defense of the Triplet Loss for Person Re-Identification*. arXiv:1703.07737. 2017.

[160] Weihua Chen, Xiaotang Chen, Jianguo Zhang, and Kaiqi Huang. "Beyond Triplet Loss: A Deep Quadruplet Network for Person Re-Identification". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[161] M. Köstinger, M. Hirzer, P. Wohlhart, P. M. Roth, and H. Bischof. "Large Scale Metric Learning from Equivalence Constraints". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[162] Shengcai Liao, Yang Hu, Xiangyu Zhu, and Stan Z. Li. "Person Re-Identification by Local Maximal Occurrence Representation and Metric Learning". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[163] Hailin Shi, Yang Yang, Xiangyu Zhu, Shengcai Liao, Zhen Lei, Weishi Zheng, and Stan Z. Li. "Embedding Deep Metric for Person Re-identification: A Study Against Large Variations". In: *European Conference on Computer Vision (ECCV)*. Springer International Publishing, 2016, pp. 732–748.

[164] Jiaolong Yang, Peiran Ren, Dongqing Zhang, Dong Chen, Fang Wen, Hongdong Li, and Gang Hua. "Neural Aggregation Network for Video Face Recognition". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[165] Niall McLaughlin, Jesus Martinez del Rincon, and Paul Miller. "Recurrent Convolutional Network for Video-Based Person Re-Identification". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[166] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices". In: *arXiv preprint arXiv:1707.01083* (2017).

[167] Yichao Yan, Bingbing Ni, Zhichao Song, Chao Ma, Yan Yan, and Xiaokang Yang. "Person Re-identification via Recurrent Feature Aggregation". In: *European Conference on Computer Vision (ECCV)*. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Springer International Publishing, 2016.

[168] Lin Chen, Hua Yang, Ji Zhu, Qin Zhou, Shuang Wu, and Zhiyong Gao. "Deep Spatial-Temporal Fusion Network for Video-Based Person Re-Identification". In: *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW*. 2017.

[169] Shuangjie Xu, Yu Cheng, Kang Gu, Yang Yang, Shiyu Chang, and Pan Zhou. "Jointly Attentive Spatial-Temporal Pooling Networks for Video-based Person Re-Identification". In: *International Conference on Computer Vision (ICCV)*. 2017.

[170] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. "3D Object Representations for Fine-Grained Categorization". In: *International Conference on Computer Vision (ICCV) Workshops*. 2013.

[171] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. "A Large-Scale Car Dataset for Fine-Grained Categorization and Verification". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[172] Jakub Špaňhel, Jakub Sochor, Roman Juránek, Adam Herout, Lukáš Maršík, and Pavel Zemčík. "Holistic Recognition of Low Quality License Plates by CNN Using Track Annotated Data". In: *Conference on Advanced Video and Signal Based Surveillance (AVSS)*. 2017.

[173] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *arXiv preprint arXiv:1602.07261* (2016).

[174] D Kinga and J Ba Adam. "A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*. 2015.

[175] Zheng Tang, Milind Naphade, Ming-Yu Liu, Xiaodong Yang, Stan Birchfield, Shuo Wang, Ratnesh Kumar, David Anastasiu, and Jenq-Neng Hwang. "CityFlow: A City-Scale Benchmark for Multi-Target Multi-Camera Vehicle Tracking and Re-Identification". In: *arXiv preprint arXiv:1903.09254* (2019).

[176] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "Facenet: A Unified Embedding for Face Recognition and Clustering". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[177] Thorsten Litfin and Gerd Wolfram. "New Automated Checkout Systems". In: *Retailing in the 21st Century: Current and Future Trends*. Ed. by Manfred Krafft and Murali K. Mantrala. 2010.

[178] Milind Naphade, Shuo Wang, David C Anastasiu, Zheng Tang, Ming-Ching Chang, Yue Yao, Liang Zheng, Sharifur Rahman, et al. "The 6th AI City Challenge". In: *Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*. 2022.

[179] Gavin Chappell, David Durdan, Greg Gilbert, Lyle Ginsburg, Jeff Smith, and Joseph Tobolski. "Auto-ID in the Box: the Value of Auto-ID Technology in Retail Stores". In: *Auto-ID Center* (2003).

[180] Matthias Hauser, Sebastian A Günther, Christoph M Flath, and Frédéric Thiesse. "Towards Digital Transformation in Fashion Retailing: A Design-oriented IS Research Study of Automated Checkout Systems". In: *Business & Information Systems Engineering* (2019).

[181] Matthias Hauser, Sebastian Günther, Christoph Flath, and Frédéric Thiesse. "Leveraging RFID Data Analytics for the Design of an Automated Checkout System". In: (2017).

[182] MFM Busu, I Ismail, MF Saaid, and SM Norzeli. "Auto-checkout System for Retails Using Radio Frequency Identification (RFID) Technology". In: *Control and System Graduate Research Colloquium*. 2011.

[183] Matthew Ritchie, Tiana Longino, Daniel Garza, and Alp Katranci. "Optimized Automated Checkout Process for Major Food Retailers". In: (2021).

[184] Yesenia Aquilina and Michael A Saliba. "An Automated Supermarket Checkout System Utilizing a SCARA Robot: Preliminary Prototype Development". In: *Procedia Manufacturing* (2019).

[185] Namitha James, Nikhitha Theresa Antony, Sara Philo Shaji, Sherin Baby, and Jyotsna Annakutty. "Automated Checkout for Stores: A Computer Vision Approach". In: *Revisita Geintec-Gestao Inovacao E Tecnologias* (2021).

[186] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection". In: *arXiv preprint arXiv:1506.02460* (2015).

[187] Chengjian Feng, Yujie Zhong, Yu Gao, Matthew R Scott, and Weilin Huang. "TOOD: Task-aligned One-stage Object Detection". In: *International Conference on Computer Vision (ICCV)*. 2021.

[188] Qiang Chen, Yingming Wang, Tong Yang, Xiangyu Zhang, Jian Cheng, and Jian Sun. "You Only Look One-level Feature". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021.

[189] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: (1960).

[190] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. "ByteTrack: Multi-Object Tracking by Associating Every Detection Box". In: *arXiv preprint arXiv 2110.06864* (2021).

[191] Lingxiao He, Xingyu Liao, Wu Liu, Xinchen Liu, Peng Cheng, and Tao Mei. "FastReID: A Pytorch Toolbox for General Instance Re-Identification". In: *arXiv preprint arXiv:2006.02631* (2020).

[192] Jiang-Jiang Liu, Qibin Hou, Ming-Ming Cheng, Changhu Wang, and Jiashi Feng. "Improving convolutional networks with self-calibrated convolutions". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[193] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. "Deep high-resolution representation learning for visual recognition". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2020).

[194] Christian Zimmermann, Duygu Ceylan, Jimei Yang, Bryan Russell, Max Argus, and Thomas Brox. "Freihand: A Dataset for Markerless Capture of Hand Pose and Shape from Single RGB Images". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[195] Gyeongsik Moon, Shoou-I Yu, He Wen, Takaaki Shiratori, and Kyoung Mu Lee. "Interhand2. 6m: A Dataset and Baseline for 3D Interacting Hand Pose Estimation from a Single RGB Image". In: *European Conference on Computer Vision (ECCV)*. 2020.

[196] Junjie Huang, Zheng Zhu, Feng Guo, and Guan Huang. "The Devil is in the Details: Delving Into Unbiased Data Processing for Human Pose Estimation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[197] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. "PointRend: Image Segmentation as Rendering". In: *arXiv preprint arXiv:1912.08193* (2019).

[198] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. "Hybrid Task Cascade for Instance Segmentation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[199] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. "Generative image inpainting with contextual attention". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[200] Shengyu Zhao, Jonathan Cui, Yilun Sheng, Yue Dong, Xiao Liang, Eric I Chang, and Yan Xu. "Large Scale Image Completion via Co-modulated Generative Adversarial Networks". In: *arXiv preprint arXiv:2103.10428* (2021).

[201] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. "MMDetection: Open MMLab Detection Toolbox and Benchmark". In: *arXiv preprint arXiv:1906.07155* (2019).

[202] Rui Liu, Hanming Deng, Yangyi Huang, Xiaoyu Shi, Lewei Lu, Wenxiu Sun, Xiaogang Wang, Jifeng Dai, and Hongsheng Li. "FuseFormer: Fusing Fine-Grained Information in Transformers for Video Inpainting". In: *International Conference on Computer Vision (ICCV)*. 2021.

[203] Yanhong Zeng, Jianlong Fu, and Hongyang Chao. "Learning Joint Spatial-Temporal Transformations for Video Inpainting". In: *European Conference on Computer Vision (ECCV)*. 2020.

[204] Zoran Zivkovic and Ferdinand van der Heijden. "Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction". In: *Pattern Recognition Letters* (2006).

[205] Yue Yao, Liang Zheng, Xiaodong Yang, Milind Napthade, and Tom Gedeon. "Attribute Descent: Simulating Object-Centric Datasets on the Content Level and Beyond". In: *arXiv preprint arXiv:2202.14034* (2022).

[206] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. "Places: A 10 million Image Database for Scene Recognition". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2017).

[207] Chengjian Feng, Yujie Zhong, Yu Gao, Matthew R. Scott, and Weilin Huang. *TOOD: Task-aligned One-stage Object Detection*. 2021.

[208] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. *MMDetection: Open MMLab Detection Toolbox and Benchmark*. 2019.

[209] Christian Herdtweck and Christian Wallraven. "Estimation of the Horizon in Photographed Outdoor Scenes by Human and Machine". In: *PLoS ONE* (2013).

[210] Guillaume A. Rousselet, Olivier R. Joubert, and Michele Fabre-Thorpe. "How Long to Get to the "Gist" of Real-world Natural Scenes?" In: *Visual Cognition* (2005).

[211] Kirsti Andersen. *Brook Taylor's Work on Linear Perspective*. Springer New York, 1992.

[212] Jana Košecká and Wei Zhang. "Video compass". In: *European Conference on Computer Vision (ECCV)*. 2002.

[213] Bruno Caprile and Vincent Torre. "Using Vanishing Points for Camera Calibration". In: *International Journal of Computer Vision (IJCV)* (1990).

[214] Radu Orghidan, Joaquim Salvi, Mihaela Gordan, and Bogdan Orza. "Camera Calibration Using Two or Three Vanishing Points". In: *Federated Conference on Computer Science and Information Systems (FedCSIS)*. 2012.

[215] Shiyao Huang, Xianghua Ying, Jiangpeng Rong, Zeyu Shang, and Hongbin Zha. "Camera Calibration From Periodic Motion of a Pedestrian". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[216] Lucas Teixeira, Fabiola Maffra, and Atta Badii. "Scene Understanding for Auto-Calibration of Surveillance Cameras". In: 2014.

[217] Jingchen Liu, Robert T. Collins, and Yanxi Liu. "Surveillance Camera Autocalibration based on Pedestrian Height Distribution". In: *British Machine Vision Conference (BMVC)* (2011).

[218] Sung Chun Lee and Ran Nevatia. "Robust Camera Calibration Tool for Video Surveillance Camera in Urban Environment". In: *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2011.

[219] Jonathan Deutscher, Michael Isard, and John MacCormick. “Automatic Camera Calibration from a Single Manhattan Image”. In: *European Conference on Computer Vision (ECCV)*. 2002.

[220] Horst Wildenauer and Allan Hanbury. “Robust Camera Self-Calibration from Monocular Images of Manhattan Worlds”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[221] M. Dubská, A. Herout, R. Juránek, and J. Sochor. “Fully Automatic Roadside Camera Calibration for Traffic Surveillance”. In: *Transactions on Intelligent Transportation Systems (T-ITS)* (2015).

[222] Derek Hoiem, Alexei A. Efros, and Martial Hebert. “Putting Objects in Perspective”. In: *International Journal of Computer Vision (IJCV)* (2008).

[223] Patrick Wang, Kenneth Morton, Peter Torrione, and Leslie Collins. “Viewpoint Adaptation for Rigid Object Detection”. In: *arXiv preprint arXiv:1702.07451* (2017).

[224] Menghua Zhai, Scott Workman, and Nathan Jacobs. “Detecting Vanishing Points using Global Image Context in a Non-Manhattan World”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[225] Jose Lezama, Rafael Grompone von Gioi, Gregory Randall, and Jean-Michel Morel. “Finding Vanishing Points via Point Alignments in Image Primal and Dual Domains”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[226] Nathan Jacobs, Mohammad T. Islam, and Scott Workman. “Cloud Motion as a Calibration Cue”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013.

[227] Scott Workman, Menghua Zhai, and Nathan Jacobs. “Horizon Lines in the Wild”. In: *British Machine Vision Conference (BMVC)*. 2016.

[228] Olga Barinova, Victor Lempitsky, Elena Tretiak, and Pushmeet Kohli. “Geometric Image Parsing in Man-Made Environments”. In: *European Conference on Computer Vision (ECCV)*. 2010.

[229] Patrick Denis, James H. Elder, and Francisco J. Estrada. “Efficient Edge-Based Methods for Estimating Manhattan Frames in Urban Imagery”. In: *European Conference on Computer Vision (ECCV)*. 2008.

[230] Roee Litman, Simon Korman, Alex Bronstein, and Shai Avidan. “Inverting RANSAC: Global model detection via inlier rate estimation”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[231] Nobuyuki Otsu. “A Threshold Selection Method from Gray-level Histograms”. In: *Automatica* (1975).

[232] Shuai Yi, Hongsheng Li, and Xiaogang Wang. “Understanding Pedestrian Behaviors from Stationary Crowd Groups”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[233] Tao Yang, Quan Pan, Jing Li, and S. Z. Li. “Real-time Multiple Objects Tracking with Occlusion Handling in Dynamic Scenes”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2005.

[234] Pierre Moulon, Pascal Monasse, and Renaud Marlet. “Global Fusion of Relative Motions for Robust, Accurate and Scalable Structure from Motion”. In: *International Conference on Computer Vision (ICCV)*. 2013.

Appendix A

Teasers

The teasers used in the papers to shortly summarize single methods are placed here.

A.1 LandmarksCalib

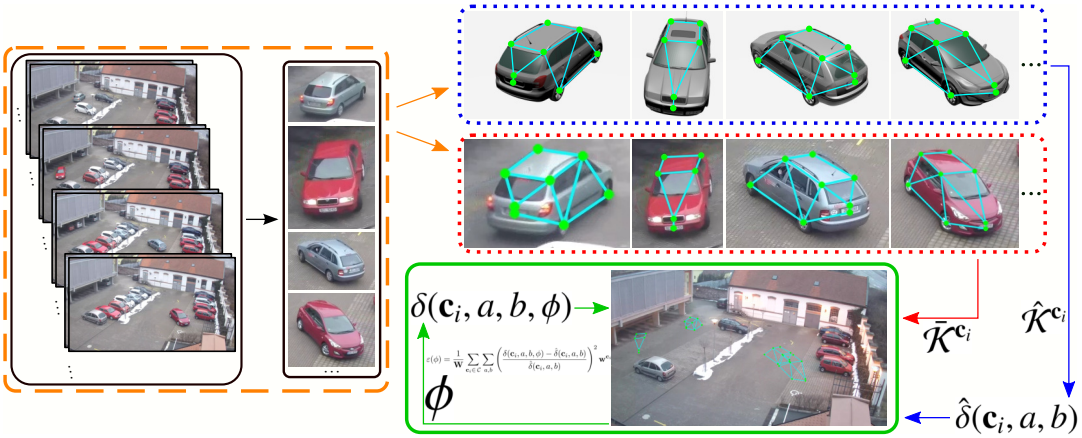

FIGURE A.1: Overview of the proposed approach. Vehicles observed in the input video (dashed – orange) are classified (to obtain the exact make & model) and processed by a landmark detector (middle dotted – red). For visible landmarks $\bar{\mathcal{K}}^{\mathbf{c}_i}$, their 3D positions $\hat{\mathcal{K}}^{\mathbf{c}_i}$ are obtained from a CAD model (top dotted – blue). Pairwise distances from the known 3D positions $\hat{\delta}(\mathbf{c}_i, a, b)$ are compared with the observed 3D distances $\delta(\mathbf{c}_i, a, b, \phi)$ and the camera model ϕ is optimized (solid – green) by a global optimization method to obtain the best solution.

A.2 PlaneCalib

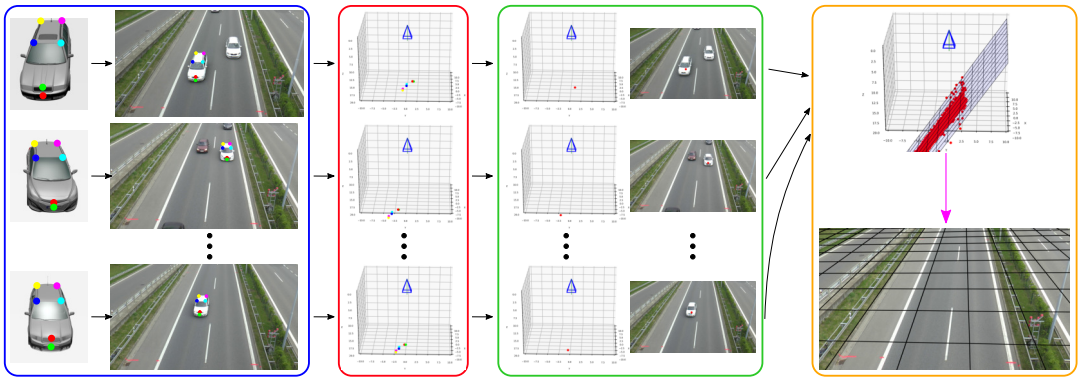

FIGURE A.2: Overview of the proposed *PlaneCalib* method. First of all, vehicles are detected, localized, classified in images (video frames), and landmarks are localized within these detections. Correspondences of the localized 2D and 3D landmarks (**blue**) are used to solve *PnP* to obtain the rotation and translation of the object with respect to the camera (**red**). For each object, the origin point which lies in the ground plane is transformed into the camera coordinate system (**green**). Finally, these potential ground plane points of all the objects observed are used to compute the calibration ground plane as the best fit to these origin points (**orange**) and the transformation of this ground plane to the world coordinate system is computed (**magenta**).

A.3 OptInOpt

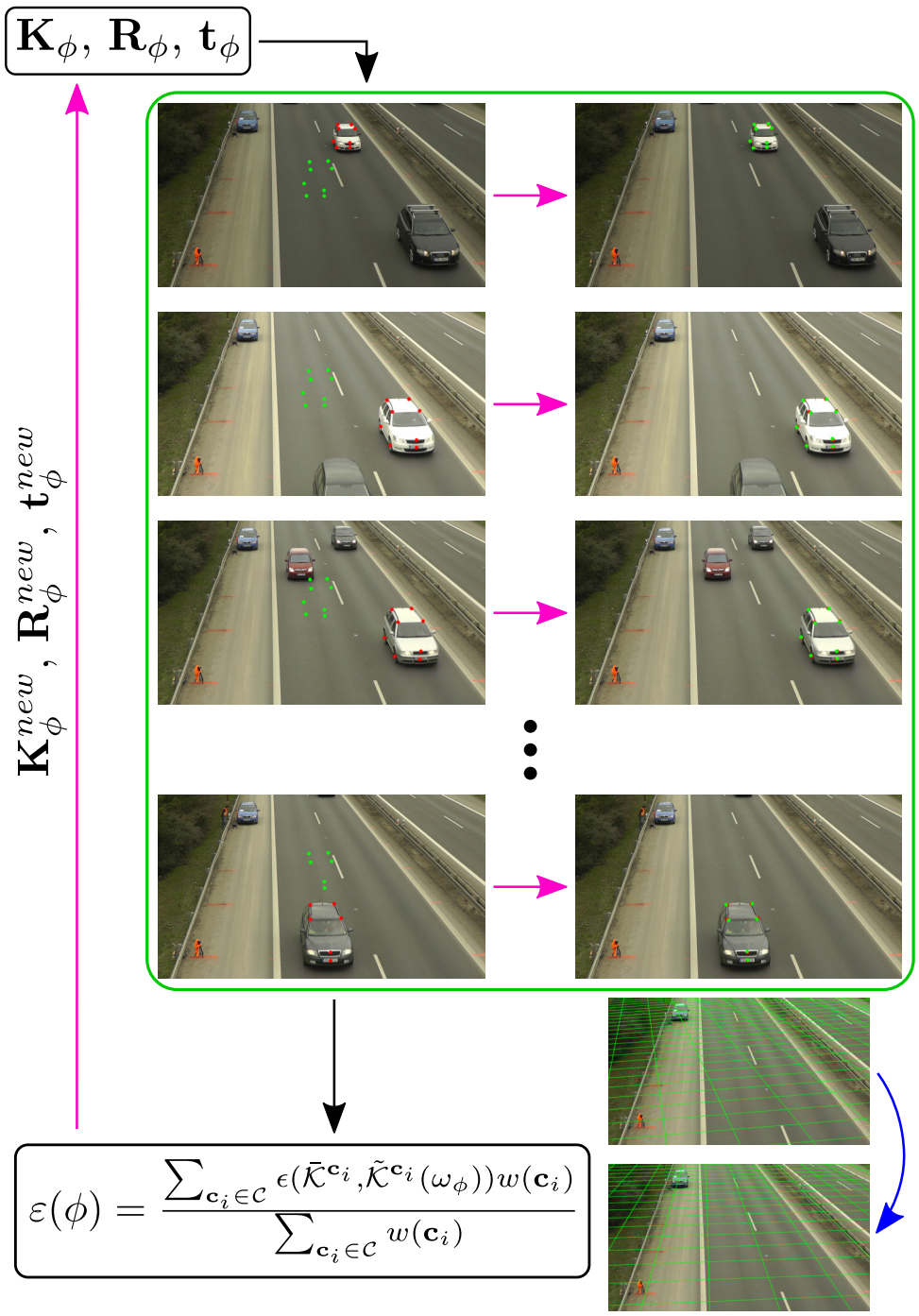

FIGURE A.3: Overview of the proposed *OptInOpt* method. In each step of optimizing the calibration parameters $\mathbf{K}_\phi$, $\mathbf{R}_\phi$, $\mathbf{t}_\phi$ (**outer magenta arrow**), optimization of individual vehicles' position is used (**green box**). Each single vehicle position is optimized (**inner magenta arrows**) based on localized landmarks and re-projection error of these landmarks w.r.t. actual calibration parameters $\mathbf{K}_\phi$, $\mathbf{R}_\phi$, $\mathbf{t}_\phi$. All vehicles' positions are used to evaluate the objective function (**bottom black box** — see Section 3.5.3 for details). Calibration grids (blue arrows) show improvement of calibration parameters based on reduction of the objective function between optimization steps (only for visualization).